UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MASTER OF SCIENCE IN SIGNAL THEORY AND COMMUNICATIONS

Signal Processing and Machine Learning for Big Data

MASTER THESIS

Development of a text summarization system using deep neural networks

EMRULLAH ERGUN

2021

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

Título: **Development of a text summarization system using deep neural networks**

Autor: ERGUN Emrullah

Tutor: **Prof. Luis A. Hernández Gómez**

Departamento: Señales, Sistemas y Radiocomunicaciones

MIEMBROS DEL TRIBUNAL

Presidente: D. ……

Vocal: D. ……..

Secretario: D. ……..

Suplente: D. ……..

Los miembros del tribunal arriba nombrados acuerdan otorgar la calificación de: ……

Madrid, a de de 20…

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR

DE INGENIEROS DE TELECOMUNICACIÓN

MASTER OF SCIENCE IN SIGNAL THEORY AND COMMUNICATIONS

Signal Processing and Machine Learning for Big Data

MASTER THESIS

Development of a text summarization system using deep neural networks

ERGUN Emrullah

2021

RESUMEN

El tratamiento de grandes cantidades de datos textuales requiere el uso de sistemas eficaces. Los sistemas de resumen automático son capaces de abordar esta cuestión. Por lo tanto, resulta esencial trabajar en el diseño de los sistemas de resumen automático existentes e innovarlos para que sean capaces de satisfacer las demandas de datos en continuo aumento, en función de las necesidades de los usuarios. Este estudio tiende a hacer un sondeo de la literatura científica para obtener información y conocimientos sobre las investigaciones recientes en materia de resumen automático de textos. Se ha presentado una revisión de varios modelos de resumen abstracto basados en redes neuronales. El propósito de esta investigación es proporcionar una comprensión general y familiarizarse con los elementos de los recientes modelos de resumen de texto abstracto basados en redes neuronales, una arquitectura codificadora-decodificadora basada en transformadores que se considera el nuevo estado de la técnica. Basándonos en los conocimientos adquiridos en el estudio, utilizaremos un transformador previamente entrenado y lo ajustaremos a un conjunto de datos elegido. Además, crearemos un algoritmo básico de resumen de texto para comprender el principio más importante en el proceso de creación de dicho sistema.

SUMMARY

Dealing with vast amounts of textual data requires the use of efficient systems. Automatic summarization systems are capable of addressing this issue. Therefore, it becomes highly essential to work on the design of existing automatic summarization systems and innovate them to make them capable of meeting the demands of continuously increasing data, based on user needs. This study tends to survey the scientific literature to obtain information and knowledge about the recent research in automatic text summarization. A review of various neural networks based abstractive summarization models have been presented. The purpose of this research is to provide an overall understanding and familiarity with the elements of recent neural networks based abstractive text summarization models a transformer-based encoder-decoder architecture are found to be the new state-of-the-art. Based on the knowledge acquired from the survey, We will make use of a pre trained transformer and we will finetune it to our chosen dataset Moreover, we will create a basic text summarization algorithm to understand the most important principle in the creation process of such system

PALABRAS CLAVE

Resumen automático de texto, Redes Neuronales Profundas, Procesamiento de Lenguaje Natural, Transformers, Seq2Seq.

KEYWORDS

Automatic Text Summarization, Deep Learning, Natural Language Processing, Transformers, Seq2Seq

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION AND OBJECTIVES

1.1. INTRODUCTION

Managing massive amounts of textual data necessitates the deployment of efficient solutions. This is a problem that automatic summarization systems can solve. As a result, it is critical to work on the architecture of existing automatic summarization systems and reinvent them so that they can meet the demands of continuously rising data based on user needs. The purpose of this study is to research the scientific literature for information and knowledge regarding contemporary research in automatic text summarization. A survey of several abstractive summarization models based on neural networks has been presented. The goal of this study is to provide an overall understanding and familiarity with the elements of contemporary neural networks-based abstractive text summarization models, as well as to raise awareness of the challenges and concerns with these systems. Models using a transformer-based encoder-decoder design have been confirmed to be the new state-of-the-art. Based on the survey results, this article recommends using pre-trained language models in conjunction with neural network design for abstractive summarization tasks. Furthermore, we will attempt to develop a rudimentary text summarizing algorithm in order to fully understand the most crucial principle in the development of such systems.

1.2. OBJECTIVES

The main objectives in this Master Thesis can be summarized as follows:

- Review the state of the art in text summarization systems.
- To design, develop and evaluate a system based on deep neural networks.
- Learning how to use the T5 transformer architecture and fine tune it for our own tasks and dataset.
- Development of a basic Seq2Seq summarizer
- Development of a basic summarizer using a transformer-based architecture.

For the experimental analysis we will train and test all of our models on the reddit dataset. It contains pre-processed articles from the Reddit dataset. The dataset consists of 3,848,330 articles with an average length of 270 words for content and 28 words for summary. Features include strings: author, body, normalizedBody, content, summary, subreddit, subreddit_id. Content is used as document and summary is used as abstract.

During the development we will use Google drive for the stockage unit and Google Collaboratory for the processing unit. We need to make sure that we will use GPU resources when it is possible.

We will also research on the selection of a metric suited for our summarization tasks. We need to find a metric that can be compared to the target summarized sentence.

1.3. APPLICATION

Text Summarization is one of the main fields we can find in NLP (Natural Language Processing) nowadays as we live in a world where there is more and more information everywhere and we as a human are limited by our lifetime. We need to prioritize what we Will take the time to read

There are multiple direct and indirect applications of the Text Summarization field. Some of them include -

- **Financial research**: Investment banking firms spend large amounts of money acquiring information to drive their decision-making, including automated stock trading. Financial analysts inevitably hit a wall and are not able to read everything. Summarization systems tailored to financial documents like earning reports and financial news can help analysts quickly derive market signals from content.

- **Market Intelligence**: Automated summarization of key competitor content releases, news tracking, patent research, etc. to drive competitive advantage.

- **Pharma clinical phase intelligence**: Scalable summarization of ongoing research/clinical trials happening in a specific therapy area or domain. At any point there are thousands of such research papers being published and it's slow and cumbersome for pharma research teams to keep on top of all of it.

- **Newsletters**: Many weekly newsletters take the form of an introduction followed by a curated selection of relevant articles. Summarization would allow organizations to further enrich newsletters with a stream of summaries (versus a list of links), which can be a particularly convenient format on mobile.

1.4. WHAT IS TEXT SUMMARIZATION IN NLP?

"Automatic text summarization is the task of producing a concise and fluent summary while preserving key information content and overall meaning"

-Text Summarization Techniques: A Brief Survey, 2017

For text summarization, there are essentially two approaches:

- Extractive Summarization
- Abstractive Summarization

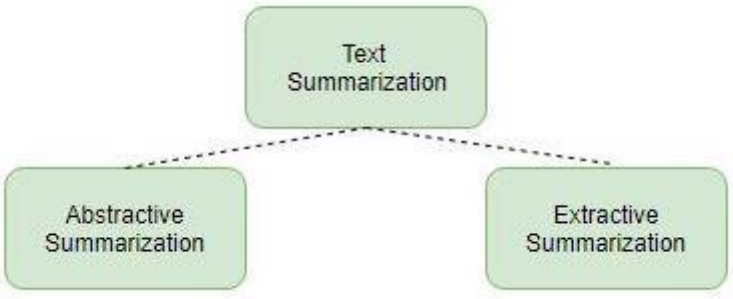

Figure 1: Overview of the task

(Image taken from: Extractive Text Summarization using NLTK in Python | The B2B News)

1.4.1. EXTRACTIVE SUMMARIZATION

The name implies what this method accomplishes. We select and extract only the key sentences or phrases from the original text. Those extracted sentences would serve as our synopsis. The graphic below depicts extractive summarization:

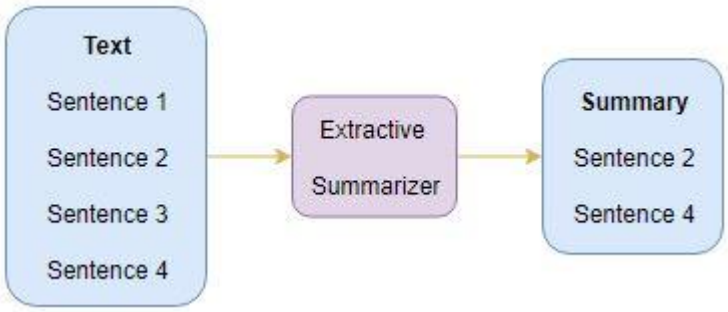

Figure 2: How does Extractive Summarizer works

(Image taken from: Extractive Text Summarization using NLTK in Python | The B2B News)

1.4.2. ABSTRACTIVE SUMMARIZATION

This is an intriguing approach. In this section, we produce new sentences based on the original material. This is in contrast to the previous extractive technique, in which we used only the sentences that were present. The sentences generated via abstractive summarization may or may not appear in the source text:

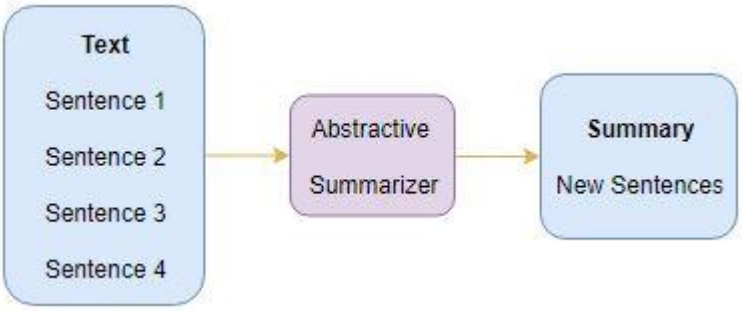

Figure 3: How does Abstractive Summarizer works

(Image taken from: Extractive Text Summarization using NLTK in Python | The B2B News)

In this study, we will be particularly interested in abstractive Summarization because the model will create sentences by itself, and this form of system is typically more robust and concentrates state-of-the-art solutions.

1.5. UNDERSTANDING KEY POINTS AND HISTORY

1.5.1. SEQUENCE-TO-SEQUENCE MODELS – A BACKDROP

In NLP, sequence-to-sequence (seq2seq) models are used to transform Type A sequences to Type B sequences. Translation of English sentences to German sentences, for example, is a sequence-to-sequence job.

Since their introduction in 2014 [1], recurrent neural network (RNN)-based sequence-to-sequence models have gained popularity. The majority of data in today's world is in the form of sequences — it could be a numerical sequence, a text series, a video frame sequence, or an audio series.

In 2015 [2], the addition of the Attention Mechanism improved the performance of these seq2seq models even further.

These sequence-to-sequence models are extremely flexible and are applied in a wide range of NLP tasks, including:

- Machine Translation
- Text Summarization
- Speech Recognition
- Question-Answering System, and so on

1.5.2. RNN BASED SEQUENCE-TO-SEQUENCE MODEL

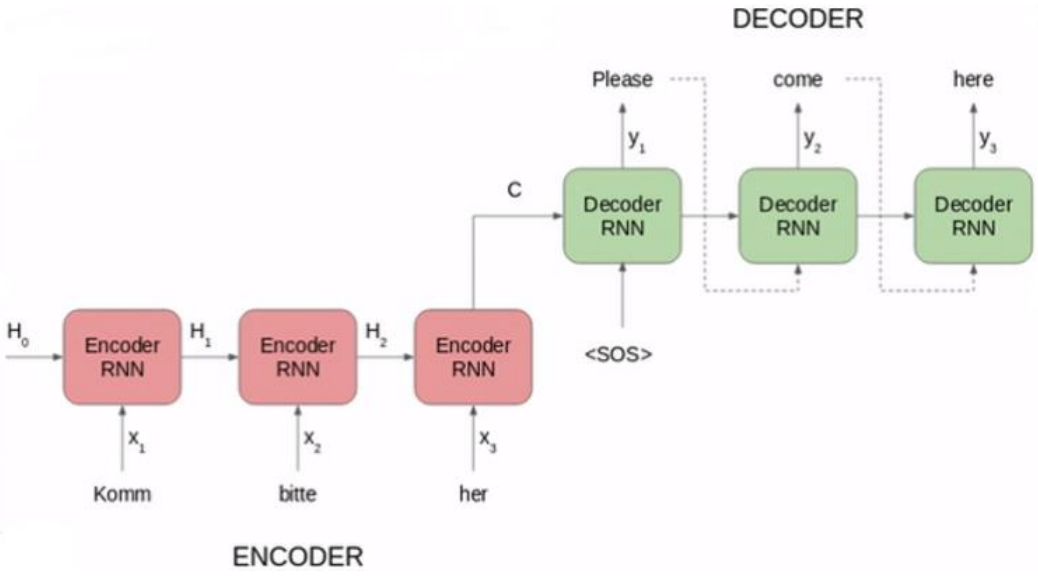

Figure 4: Architecture of a RNN based Seq2Seq model

(Image taken from: https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/)

The preceding seq2seq model is converting a German sentence to its English equivalent. Let's take it one step at a time:

- Both Encoder and Decoder are RNNs
- At every time step in the Encoder, the RNN accepts a word vector (x_i) from the input sequence and a hidden state (H_i) from the previous time step at each time step
- The hidden state is updated at each time step.
- The context vector is the hidden state from the previous unit. This section gives details on the input sequence.
- This context vector is then provided to the decoder, where it is utilized to construct the target sequence (English phrase).
- If the Attention mechanism is utilized, the weighted sum of the hidden states is provided to the decoder as the context vector.

1.5.3. CHALLENGES

Despite being so good at what it does, there are some limits to seq-2-seq models that deserve special attention:

- Dealing with long-term dependency is difficult.
- The model architecture's sequential nature prevents parallelization. Google Brain's Transformer idea addresses these issues.

1.5.4. INTRODUCTION TO THE TRANSFORMER

Vaswani et al. argued in a paper published from 2017 that Attention is all you need [3] — in other words, that recurrent building blocks are not required in a Deep Learning model for it to perform exceptionally well on NLP tasks. They proposed the Transformer, a new design capable of preserving the attention mechanism while processing sequences in parallel: all words simultaneously rather than word by word.

This architecture has eliminated the last of the two issues noted above, namely that sequences must be handled sequentially, incurring a high computational cost. Parallelism has become a reality thanks to Transformers.

Transformer-based architectures come in a variety of varieties, as we'll see in the next section. Researchers and engineers have experimented extensively and brought about innovations based on the traditional Transformer architecture. The original Transformer architecture, on the other hand, looked like this:

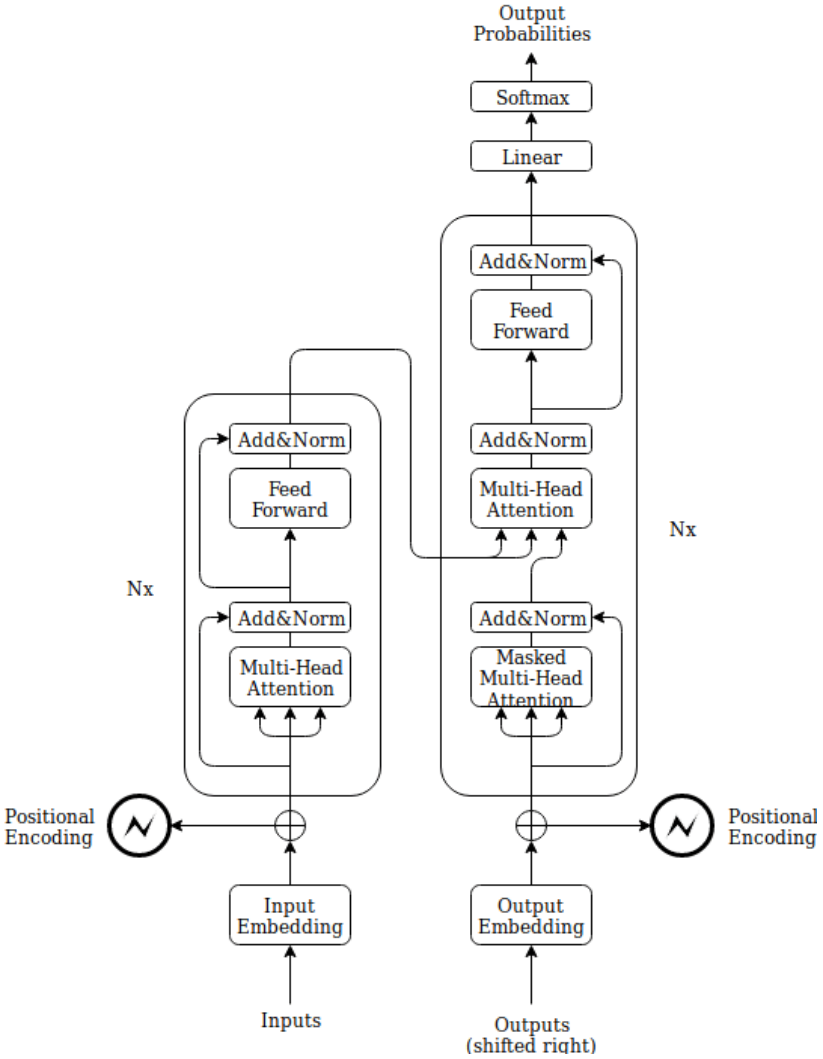

Figure 5: Architecture of the first Transformer proposed

(Image taken from: arXiv:1807.11605v1 [cs.CL] 30 Jul 2018)

Now, pay attention to the image below. The Encoder block is composed of one layer of Multi-Head Attention followed by another layer of Feed Forward Neural Network. In contrast, the decoder has an additional Masked Multi-Head Attention.

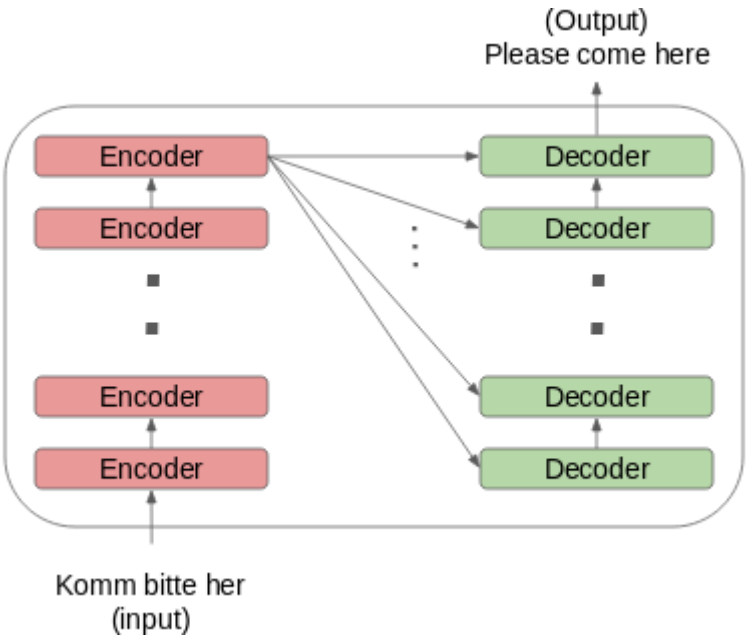

Figure 6: Stacking of unit inside the transformer

(Image taken from: https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/)

Let's take a look at how this encoder and decoder stack configuration works:

- The word embeddings of the input sequence are provided to the first encoder, where they are transformed and propagated to the next encoder.

- As indicated in the figure below, the output from the last encoder in the encoder-stack is transmitted to all of the decoders in the decoder-stack:

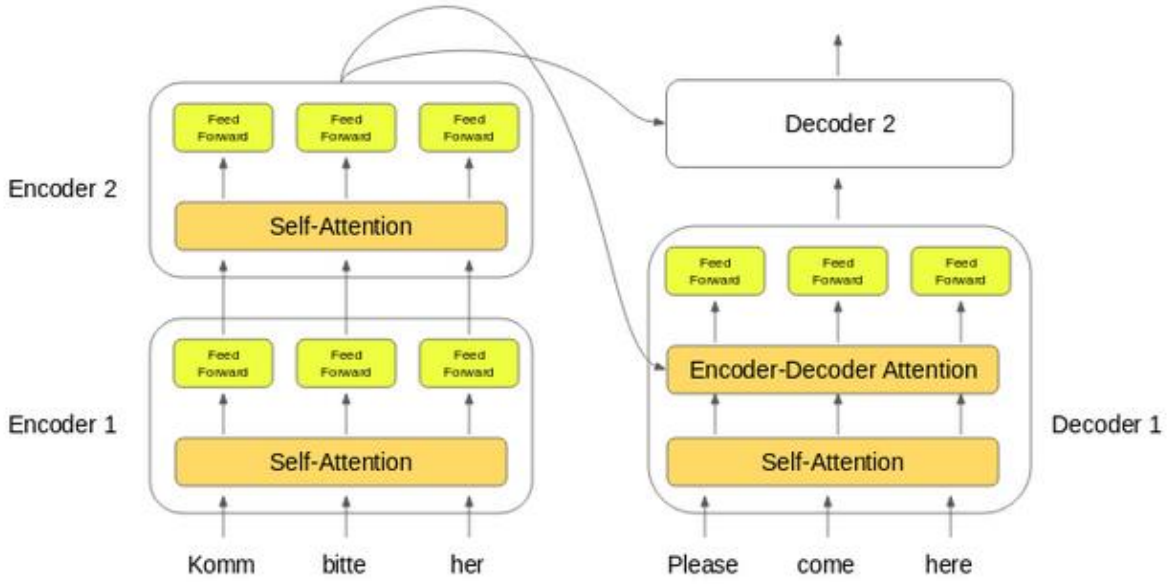

Figure 7: Staking inside the transformer opened

(Image taken from: https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/)

It's worth mentioning that, in addition to the self-attention and feed-forward layers, the decoders contain one more layer of Encoder-Decoder Attention. This allows the decoder to focus on the relevant parts of the input sequence.

GETTING A HANG OF SELF-ATTENTION

According to the paper:

"Self-attention, sometimes called intra-attention, is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence."

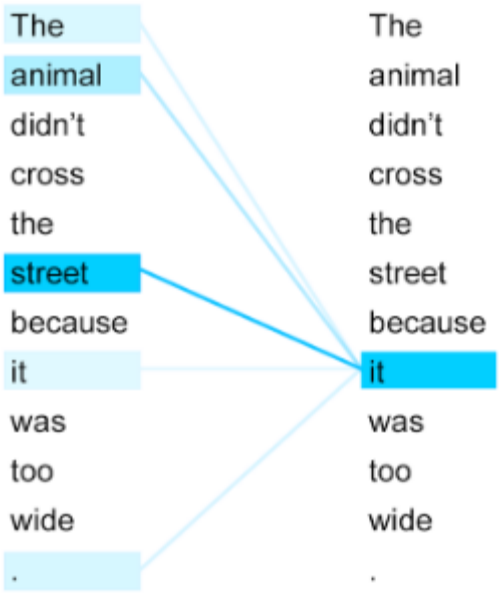

Figure 8: Attention mechanism goals

(Image taken from: https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/)

Take a look at the above picture. Can you find out what the term "it" in this statement refers to?

Is it referring to the street or the animal? It's a straightforward question for us, but not for an algorithm. When the model is processing the word "it," self-attention seeks to correlate "it" with "animal" in the same phrase.

Self-attention allows the model to examine the other words in the input sequence to gain a better grasp of a certain word in the sequence.

CALCULATING SELF-ATTENTION

According to Dontloo(2019):

*"The key/value/query concepts come from retrieval systems. For example, when you type a query to search for some video on Youtube, the search engine will map your **query** against a set of **keys** (video title, description etc.) associated with candidate videos in the database, then present you the best matched videos (**values**)."*

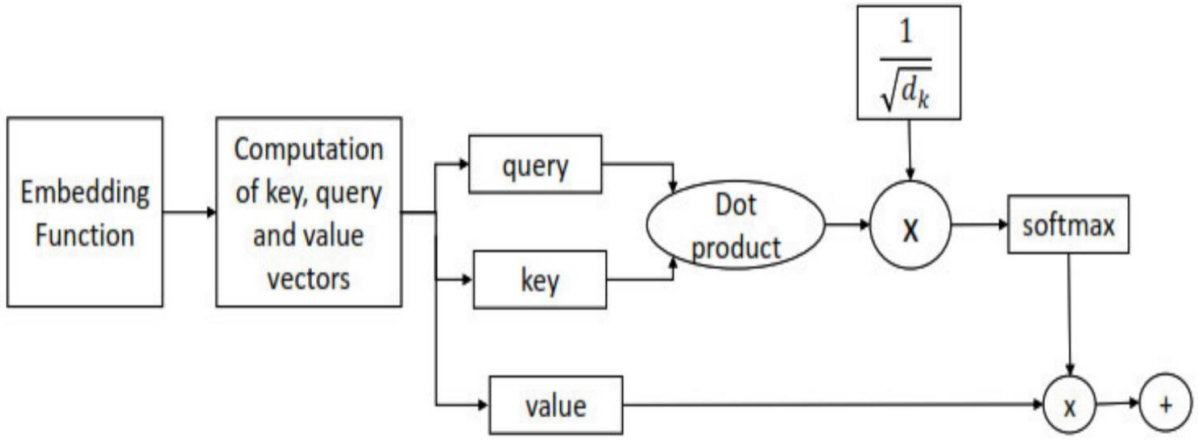

Figure 9: Construction of the attention mechanism output

The following explanation of how to calculate Self Attention vector is taken from a website. I though that it was the clearest explanation out of every paper and tutorials I read on the internet.

https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/

1. To begin, we must generate three vectors from each of the encoder's input vectors:

    1. Query Vector
    2. Key Vector
    3. Value Vector.

During the training process, these vectors are trained and updated.

2. Next, we will calculate self-attention for every word in the input sequence

3.Consider the following phrase: "Action gets results." To measure self-attention for the first word "Action," we will compute scores for each word in the sentence in regard to "Action." When we encode a certain word in an input sequence, this score influences the weight of other words.

1. The score for the first word is calculated by taking the dot product of the Query vector (q1) with the keys vectors (k1, k2, k3) of all the words:

Word	q vector	k vector	v vector	score
Action	q_1	k_1	v_1	$q_1 . k_1$
gets		k_2	v_2	$q_1 . k_2$
results		k_3	v_3	$q_1 . k_3$

2. Then, these scores are divided by 8 which is the square root of the dimension of the key vector:

Word	q vector	k vector	v vector	score	score / 8
Action	q_1	k_1	v_1	$q_1 . k_1$	$q_1 . k_1 / 8$
gets		k_2	v_2	$q_1 . k_2$	$q_1 . k_2 / 8$
results		k_3	v_3	$q_1 . k_3$	$q_1 . k_3 / 8$

3. Next, these scores are normalized using the SoftMax activation function:

Word	q vector	k vector	v vector	score	score / 8	Softmax
Action	q_1	k_1	v_1	$q_1 . k_1$	$q_1 . k_1 / 8$	x_{11}
gets		k_2	v_2	$q_1 . k_2$	$q_1 . k_2 / 8$	x_{12}
results		k_3	v_3	$q_1 . k_3$	$q_1 . k_3 / 8$	x_{13}

4. These normalized scores are then multiplied by the value vectors (v1, v2, v3) and sum up the resultant vectors to arrive at the final vector (z1). This is the output of the self-attention layer. It is then passed on to the feed-forward network as input:

Word	q vector	k vector	v vector	score	score / 8	Softmax	Softmax * v	Sum
Action	q_1	k_1	v_1	$q_1 . k_1$	$q_1 . k_1 / 8$	x_{11}	$x_{11} * v_1$	z_1
gets		k_2	v_2	$q_1 . k_2$	$q_1 . k_2 / 8$	x_{12}	$x_{12} * v_2$	
results		k_3	v_3	$q_1 . k_3$	$q_1 . k_3 / 8$	x_{13}	$x_{13} * v_3$	

So, z1 is the self-attention vector for the first word of the input sequence “Action gets results”. We can get the vectors for the rest of the words in the input sequence in the same fashion:

Word	q vector	k vector	v vector	score	score / 8	Softmax	Softmax * v	Sum#
Action		k_1	v_1	$q_2 . k_1$	$q_2 . k_1 / 8$	x_{21}	$x_{21} * v_1$	
gets	q_2	k_2	v_2	$q_2 . k_2$	$q_2 . k_2 / 8$	x_{22}	$x_{22} * v_2$	z_2
results		k_3	v_3	$q_2 . k_3$	$q_2 . k_3 / 8$	x_{23}	$x_{23} * v_3$	

Word	q vector	k vector	v vector	score	score / 8	Softmax	Softmax * v	Sum#
Action		k_1	v_1	$q_3 . k_1$	$q_3 . k_1 / 8$	x_{31}	$x_{31} * v_1$	
gets		k_2	v_2	$q_3 . k_2$	$q_3 . k_2 / 8$	x_{32}	$x_{32} * v_2$	
results	q_3	k_3	v_3	$q_3 . k_3$	$q_3 . k_3 / 8$	x_{33}	$x_{33} * v_3$	z_3

The paper improved the self-attention layer by including a mechanism known as "multi-headed" attention. This increases the attention layer's performance in two ways:

- It increases the model's capacity to focus on various positions. Yes, in the above example, z1 contains a small amount of each other encoding, but it might be dominated by the actual word itself. It would be good to know which term "it" refers to when translating a statement like "The animal didn't cross the street because it was too tired."
- It provides the attention layer with a number of "representation subspaces." As we'll see later, with multi-headed attention, we have several sets of Query/Key/Value weight matrices (the Transformer has eight attention heads, so we have eight sets for each encoder/decoder). Each of these sets is generated at random. The input embeddings (or vectors from lower encoders/decoders) are then projected onto a separate representation subspace using each set after training.

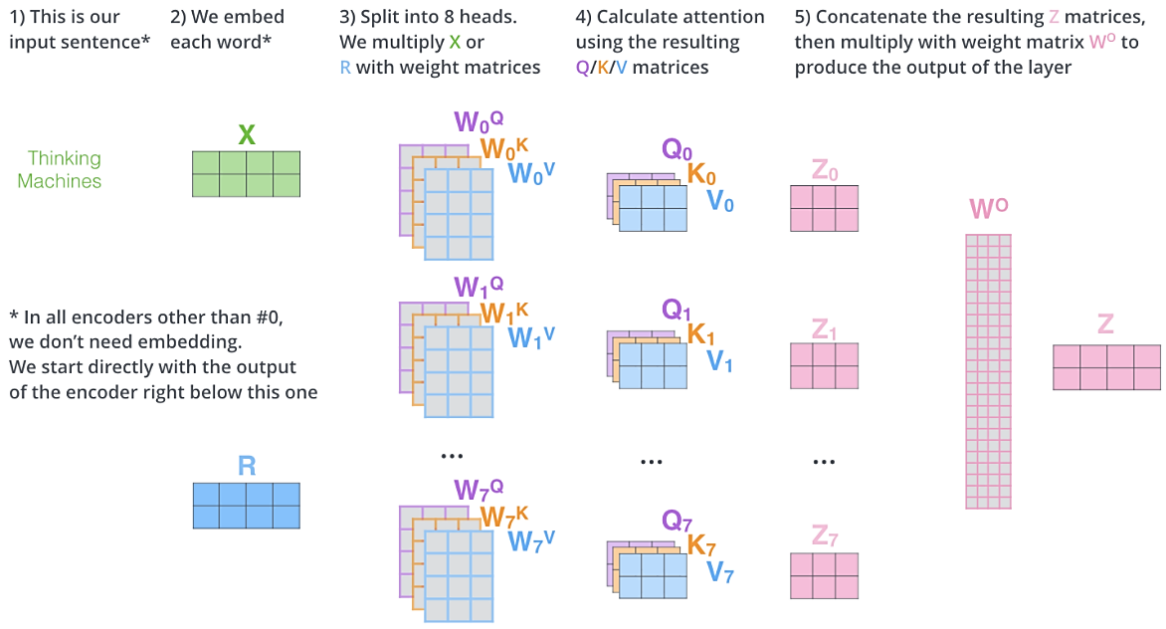

Figure 10: Multi-Head Attention pipeline

(Image taken from: https://jalammar.github.io/illustrated-transformer/)

According to the paper "Attention Is All You Need":

"Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions."

LIMITATIONS OF THE TRANSFORMER

The research was able to deal with very long phrases with many dependencies thanks to the transformer design, however there are several drawbacks we discovered after examining it:

- Attention can only deal with text strings that have a fixed length. Before being fed into the system as input, the text must be divided into a specified number of segments or chunks.
- Text chunking promotes context fragmentation. When a phrase is split down the middle, for example, a considerable quantity of context is lost. To put it another way, the text is separated without regard for the phrase or any other semantic barrier.

So, how do we cope with these errors? For this end, the Transformer-XL has been proposed.

UNDERSTANDING TRANSFORMER-XL

Transformer architectures can learn longer-term dependency. However, they can't stretch beyond a certain level due to the use of fixed-length context (input text segments). A new architecture was proposed to overcome this shortcoming in the paper – Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. [4]

In this architecture, the hidden states obtained in previous segments are reused as a source of information for the current segment. It enables modeling longer-term dependency as the information can flow from one segment to the next.

1.5.5. USING TRANSFORMER FOR LANGUAGE MODELING

Al-Rfou et al. (2018) [5] proposed the idea of applying the Transformer model for language modeling. As per the paper, the entire corpus can be split into fixed-length segments of manageable sizes. Then, we train the Transformer model on the segments independently, ignoring all contextual information from previous segments:

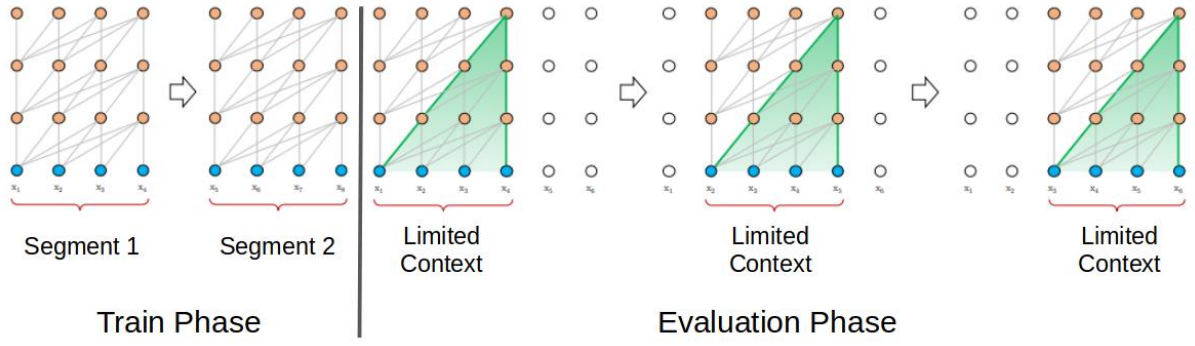

Figure 11: Model with a segment length of 4 [6]

(Image taken from: https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/)

This architecture doesn't suffer from the problem of vanishing gradients. But we have two shortcomings with this idea:

- Context-dependence is restricted since the maximum dependency distance between characters is restricted to the length of the input. The model, for example, cannot "employ" a term that appeared many sentences earlier.
- Context fragmentation — For texts longer than 512 characters, each part of that length is learned from scratch. As a result, the first tokens of each segment and between segments have no context (dependencies). This results in wasteful training and may have an impact on model performance.

1.5.6. USING TRANSFORMER-XL FOR LANGUAGE MODELING

Transformer-XL heavily relies on the vanilla Transformer (Al-Rfou et al.) [5] but introduces two innovative techniques — **Recurrence Mechanism** and **Relative Positional Encoding** —

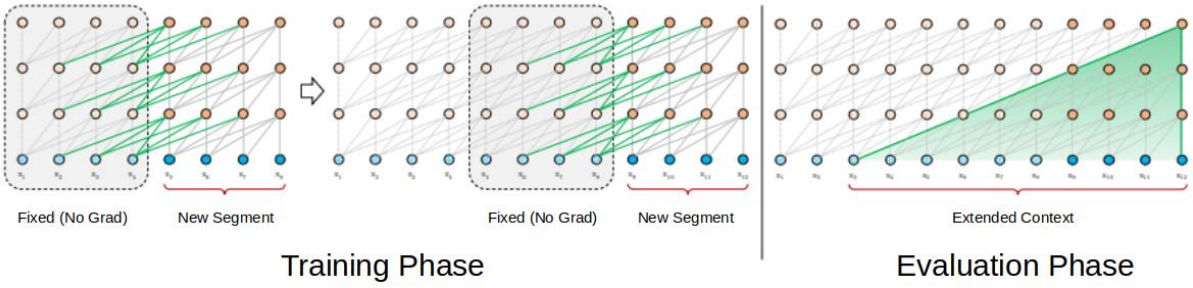

Figure 12: Transformer XL Model with a segment length of 4

(Image taken from: https://towardsdatascience.com/transformer-xl-explained-combining-transformers-and-rnns-into-a-state-of-the-art-language-model-c0cfe9e5a924)

We will only focus on the Recurrence Mechanism as it is the one that allow no context fragmentation and hence overcome the previous difficulties.

Transformer XL handles the first segment of tokens but retains the hidden layer outputs. Each hidden layer receives two inputs when the next section is processed:

- As in the vanilla version, the output of the preceding hidden layer of that segment (the grey arrows in the chart below).
- The preceding hidden layer's output from the previous segment (the green arrows) that allows the model to generate long-term dependencies.

Technically, the two inputs are concatenated and then utilized to compute the Key and Value matrices of the current segment's (current Head of the current layer of the current layer.

During the evaluation phase, the representations from the previous segments can be reused instead of being computed from scratch (as is the case of the Transformer model). This, of course, increases the computation speed manifold.

1.6. STATE OF THE ART

So far, the methodologies presented are examples of pure techniques that can be utilized to build summarization systems. The current tendency in systems is to take a hybrid approach, combining and integrating methodologies (e.g., cue phrases method combined with position and word frequency-based methods in [7], or position, length weight of sentences combined with similarity of these sentences with the headline in [8]). We will discuss the major aspects of Encoder-decoder architecture in this section

in broad lines because we have given a detailed overview of the traditional ways used in summarization and there are a large number of diverse techniques and systems.

1.6.1. ENCODER DECODER ARCHITECTURE

The selection of encoder-decoder architecture provides us with certain choices of designing our encoder and decoder with standard RNN/LSTM/GRU, bidirectional RNN/LSTM/GRU, Transformer, BERT/GPT-2 architecture, or the very recent BART model.

RECURRENT NEURAL NETWORK (RNN)

RNN architecture is particularly suited to tasks involving sequential data. [9] was the first to develop a novel network for statistical machine translation that consisted of two RNNs as encoder and decoder. This model was refined and expanded by integrating a method for neural machine translation that automatically looks for elements of the source text that indicate significance in identifying a target word (attention mechanism) [10]. Although abstractive summarization is a distinct function from machine translation, the two are closely connected because they are both sequence-to-sequence learning tasks. Many deep learning algorithms are being used to construct abstractive summaries because they are inspired by the success of neural machine translation.

LONG SHORT-TERM MEMORY

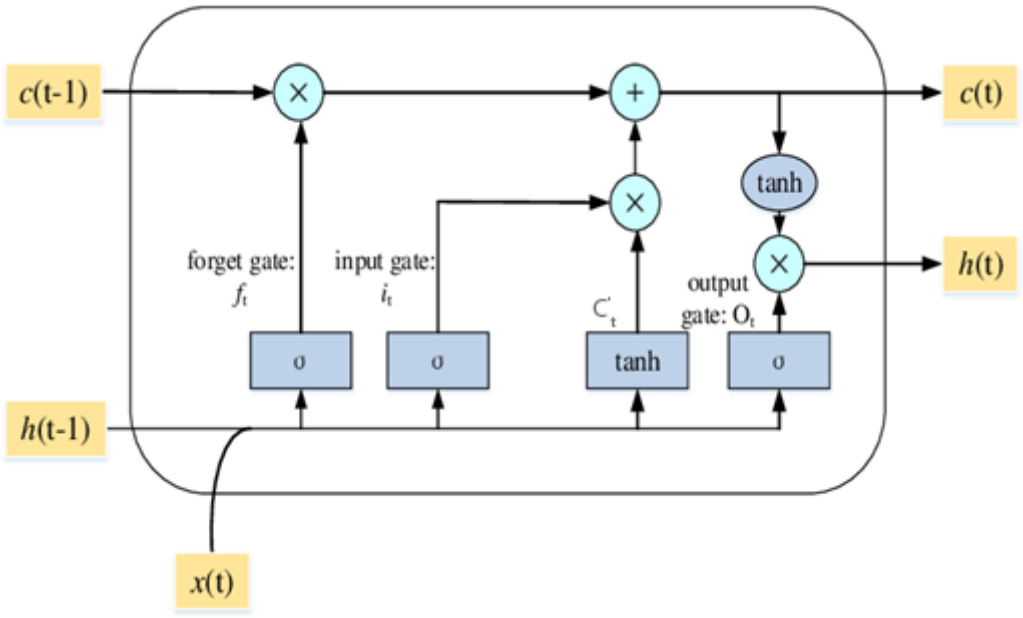

Figure 13: The architecture of an LSTM cell.

(Image taken from: https://www.researchgate.net/figure/The-structure-of-the-LSTM-unit_fig2_331421650)

The LSTM introduced by [11] is a very special variant of RNN [12] because it can deal with the problem of long-term dependencies. For example, if the next word in a sequence is to be predicted and the correctly predicted word is dependent on information mentioned a few sentences before (past information), RNN is incapable of retaining information for an extended period of time. This is where (long-term gaps/dependencies) LSTMs come into play. LSTM can retain information for longer periods

of time. The architecture of an LSTM cell is depicted in Fig. 13. In LSTM, there is a forget gate, an input gate, and an output gate that use the sigmoid activation function. The forget gate determines which information to keep and which to discard. The input gate changes the cell's state, while the output gate determines the next concealed state. (2) to (7) describe the operation of an LSTM cell:

$$f_t = \sigma\left(\omega_f \cdot [h_{t-1}, x_t] + b_f\right) \quad (2)$$

$$i_t = \sigma\left(\omega_i \cdot [h_{t-1}, x_t] + b_i\right) \quad (3)$$

$$C'_t = \tanh\left(\omega_C \cdot [h_{t-1}, x_t] + b_C\right) \quad (4)$$

$$C_t = f_t * C_{t-1} + i_t * C'_t \quad (5)$$

$$o_t = \sigma(\omega_o [h_{t-1}, x_t] + b_o) \quad (6)$$

$$h_t = o_t * \tanh(C_t) \quad (7)$$

In (2) to (7) and Fig. 13, x_t is the input, C'_t t is the candidate (holds possible values to add to the cell state), C_t is the new cell state, f_t is the output of forget gate, i_t is the output of the input gate, C_{t-1} is the previous state of the cell, h_{t-1} is the previous hidden state, h_t is the new hidden state while ω and b represent the corresponding weights and biases.

Vanishing Gradient is a fairly frequent issue in multi-layered neural networks. The more layers a neural network has, the more capacity it has, which means it can learn from larger training datasets and map more complicated functions from input to output. When the gradient is backpropagated across the network, its value steadily declines. When it gets close to the first layers, the gradient becomes noticeably smaller. As a result, the initial layers' weights and biases are not appropriately updated. Because the initial layers are crucial in detecting the pieces of incoming data, the disappearing gradient contributes to the network's overall imprecision.

GATED RECURRENT UNIT (GRU)

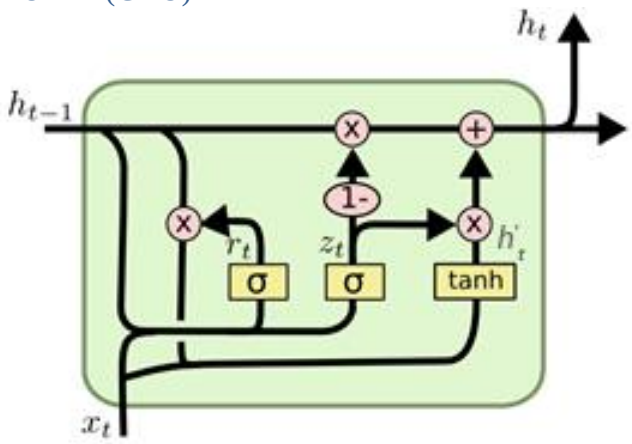

Figure 14: Architecture of a GRU cell.

(Image taken from: https://stackoverflow.com/questions/57448101/discrepancy-between-diagram-and-equations-of-gru)

GRU [13] is a variation of LSTM [11] because to resemblance in their architecture. It deals with the issue of vanishing gradients in recurrent neural networks. GRU's architecture includes an update gate and a reset gate. The update gate handles information that enters memory and assists the model in determining which previous information should be passed on, whereas the reset gate handles information that exits memory and assists the model in determining which previous knowledge the network should forget. These are the two vectors that determine what information is sent to the output. GRU's design and functions are depicted in Fig. 14. It may be stated as follows:

$$z_t = \sigma\left(\omega_z \cdot [h_{t-1}, x_t]\right) \quad (8)$$

$$r_t = \sigma\left(\omega_r \cdot [h_{t-1}, x_t]\right) \quad (9)$$

$$h'_t = \tanh\left(\omega \cdot [r_t * h_{t-1}, x_t]\right) \quad (10)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * h'_t \quad (11)$$

In (8) to (11), x_t is the input, z_t is the update gate vector, r_t is the reset gate vector, h'_t is the tanh activation vector and h_t is the output. The ω, ω_r, and ω_z are the corresponding weight vectors.

BI-DIRECTIONAL RNN/LSTM/GRU

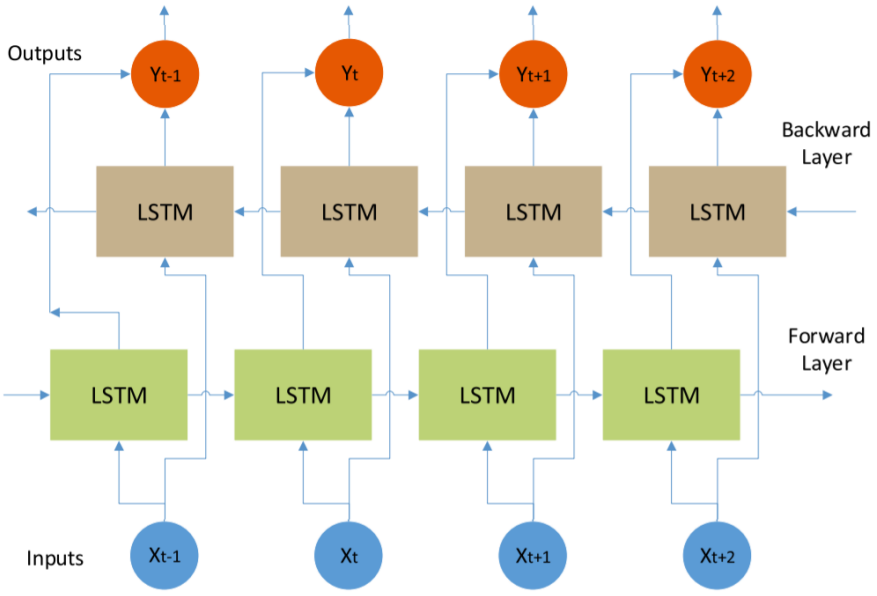

Figure 15: Bidirectional LSTM network.

(Image taken from: https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/)

Bidirectional neural networks [14] use two sequences to predict output: one in the forward direction and one in the reverse direction. It suggests that we may generate predictions of the present state utilizing information from prior time steps as well as subsequent time steps utilizing bidirectional networks. As a result, the network can collect a fuller context and solve issues more effectively. A bidirectional LSTM network is seen in Fig. 15. There are two LSTM layers: one forward and one backward. The input is sent to both the forward and backward layers. The output is a concatenation of the forward and backward layers' outputs.

TRANSFORMER

Google launched Transformers [3] in 2017 as a breakthrough for sequence learning challenges. Transformers are purely dependent on attention processes, which eliminates the necessity for both recurrent and convolution units. Its architecture consists of numerous layers of encoders and decoders. Attention units and feed-forward units make up the encoder and decoder blocks. The encoder section consists of a stack of six identical encoder units, whereas the decoder section consists of a stack of six identical decoder units. Each encoder unit contains both a multi-head attention unit and a feedforward unit. The word embeddings are propagated to the first encoder, which transforms them and passes them on to the next encoder. This procedure is repeated, and the result of Fig. 5 is the transformer design.

Figure 9 depicts the transformer's attention mechanism. The last encoder unit is moved to the decoder unit (all decoders in the stack).

The transformer's attention mechanism, seen in Fig. 9, is highly intriguing. It is required for the architecture to work. The feedforward layer receives this output. Because self-attention is computed numerous times in parallel in the transformer, it is also known as multi-head attention.

This aspect is fully handled in our research because we employed a transformer to complete our Text Summarization task.

BIDIRECTIONAL ENCODER REPRESENTATIONS FROM TRANSFORMER-GENERATIVE PRE-TRAINED TRANSFORMER (BERT-GPT)

BERT [15], released by Google in 2019, enables the application of a pre-trained language model to a variety of NLP tasks, whereas GPT [16], announced by Open AI in 2018, similarly pre-trains a language model on a vast body of text, which can subsequently be fine-tuned on a variety of specialized tasks. The main distinction between the two is that BERT can do bidirectional training, whereas GPT can only do unidirectional training. Both are transformer-based in terms of architecture. The BERT encoder is a multilayer transformer encoder, while the GPT decoder is a multilayer transformer decoder. For the first layer of GPT-2, byte pair encodings (BPE) are employed as word vectors.

BERT is divided into two modules: pretraining and fine-tuning. Masked Language Model (MLM) for bidirectional prediction and Next Sentence Prediction (NSP) for sentence-level comprehension are used to train BERT. The sum of token embeddings, segment embeddings, and position embeddings is the input embedding for BERT. In the MLM task, 15% of the words in the input sequence are substituted with a mask token, and the model predicts the masked word using the context supplied by the other words in the sequence. A classification layer is added, which receives the encoder's output. The embedding matrix is then multiplied by the output vectors. Finally, the likelihood of each vocabulary word is calculated using SoftMax.

In the NSP task, a pair of sentences is fed into the model, and the model predicts if the second sentence will appear later in the original document. A [CLS] token is placed to the beginning of the first sentence, and a [SEP] token is attached to the end of each sequence. Each token has a phrase that identifies the embedding. Every token is also given a positional embedding to denote its location in the sequence. BERT can be fine-tuned for various natural language applications by adding a layer to the fundamental model.

BIDIRECTIONAL AND AUTOREGRESSIVE TRANSFORMERS (BART)

BART, which was presented lately by [10], consists of two fundamental components: a bidirectional encoder and an autoregressive decoder. Both components are built as a sequence-to-sequence model and use a transformer-based design. During pre-training, a noise function is used to inject noise into the text, and the system learns to reconstitute the true text from the distorted text. When fine-tuned for text generation, it provides excellent performance in a variety of applications such as abstractive discourse, question answering, and text summarization [17].

The BART basic model contains 6 layers in each encoder and decoder, but the big model has 12 layers. Several strategies are used to pretrain BART, including as token masking, in which random tokens are substituted with [MASK]. Token deletion deletes specified tokens and inserts new tokens in their place. Some text regions are replaced with a [MASK] token during text-infilling. The sentences in the text are randomly mixed together in sentence permutation. In document rotation, a random token is picked to be the document's start. The section of the document preceding the insertion of the random token at the end. BART may be fine-tuned such that the representations it generates may be used by other applications such as sequence classification, token classification, sequence synthesis, and machine translation.

The researchers also compared the pre-training objectives of the GPT language model, Permuted Language model, masked language model, multitask masked language model, and masked sequence to sequence. Token masking was shown to be highly significant, left-to-right pre-training enhances Natural Language Generation (NLG) tasks, and bidirectionality is critical for question answering systems.

1.6.2. MECHANISMS

Mechanisms are capabilities that are added to the fundamental neural encoder-decoder architecture to solve specific concerns with abstractive summarization systems and improve the resulting summaries.

ATTENTION

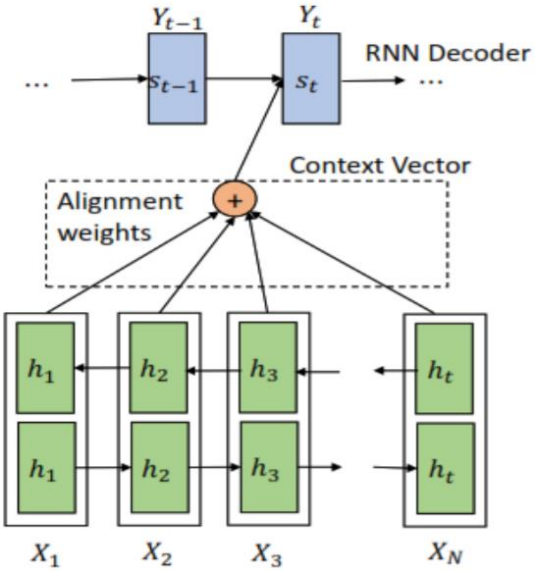

Figure 16: Attention mechanism proposed by [10].

(Image taken from: https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/)

The goal behind the attention mechanism is to focus and pay more attention to some chunks of input data than others. It was first used in neural machine translation in [10]. Later on, it was used for a variety of different tasks, including abstractive summarization. When attention is applied, the encoder's intermediate states are used to generate context vectors. When the system constructs the output sequence, it looks for context vectors that include the most important information. When attention is directed between input and output components, it is referred to as general attention; when it is directed between input components, it is referred to as self-attention. [10] presented an attention model, which is depicted in Fig. 13. The model is made up of two parts: a bidirectional RNN encoder and an RNN decoder.

As observed in Fig. 13, the encoder generates the hidden states h_1 to h_t. The context vector is computed as:

$$e_{ij} = a(s_{i-1}, h_j) \tag{12}$$

In (12), a is the alignment model.

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \tag{13}$$

The SoftMax function is used to normalize the alignment scores. The context vector is a weighted sum of a_{ij} and h_j .

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{14}$$

COPYING

When specified elements of the input sequence must be copied into the output sequence, the copying technique described in [18] can be employed. This paradigm is built around a bidirectional encoder that encodes the input sequence and a decoder that predicts the output sequence. To forecast the output sequence from the copy mode or the produce mode, a probabilistic model is utilized. Considering a set of vocabulary, $V = \{v_1, v_2, , \ldots, v_n, \}$, and an input sequence, $X = \{x_1, , x_2, \ldots, x_{Ts}\}$. There might be words in the input sequence X, that are not in V. Here, the copy mode can copy words from X that are not in V. If M is the representation of the input sequence from the encoder, s_t is the decoder state at time t and, c_t is the context, the probability of output word y_t is given by combined probabilities as

$$\begin{aligned} p\,(y_t \mid s_t, y_{t-1}, c_t, M) = p\,(y_t, g \mid s_t, y_{t-1}, c_t, M) \\ + p\,(y_t, c \mid s_t, y_{t-1}, c_t, M) \end{aligned} \tag{15}$$

where c and g are the copy and generate modes respectively.

COVERAGE

It was created to address the repetition issue in the output sequence and aid in its elimination or reduction [19]. The language that has already been covered is tracked in this model. This is accomplished by employing attention distribution to keep the system aware of the covered sequence, and the network is penalized if it attends to the same sequence many times. Mathematically, at timestep t, the coverage vector c^t is represented as

$$c^t = \sum_{t'=0}^{t-1} \alpha^{t'} \tag{16}$$

The loss term that penalizes overlap between c^t and the new attention distribution α' is expressed as

$$loss_t = \sum_i \min(\alpha_i^t, c_i^t) \tag{17}$$

POINTER-GENERATOR

This mechanism makes an attempt to propose a solution to the problem of out of vocabulary (OOV) words and factual details. It can replicate words/facts via a pointer while also being able to generate new words using a generator [19]. Together with computing an attention distribution α and a vocabulary distribution p_{vocab} , the model also calculates a generation probability denoted by p_{gen} . The generation probability represents the probability of predicting the final word either from the vocabulary or copying it from the input. Mathematically, the final probability of generating output word is expressed as,

$$p_{final}(w) = p_{gen}p_{vocab}(w) + (1 - p_{gen}) \sum_{i:w_i=w} \alpha_i \tag{18}$$

DISTRACTION

The distraction mechanism developed by [20] is to distract in such a way that it allows the reader to go between different portions of a document in order to comprehend the overall meaning for summarizing a document rather than focusing on a specific area repeatedly. The researchers applied distraction in this model from two perspectives: the first distraction task is conducted during the training process, and the second distraction task is conducted during the decoding phase. The distraction was implemented on the content vector during training by training the model not to pay too much attention to the same part again. The previously viewed vector was saved as a history content vector and blended with the vector that was now computed. Formally,

$$c_t = \tanh\left(W_c c'_t - U_c \sum_{j=1}^{t-1} c_j\right) \tag{19}$$

$$c'_t = \sum_{i=1}^{T_x} \alpha_{t,i} h_i \tag{20}$$

Where, c_j is the history content vector, $c'_t$is the input content vector, $\alpha_{t,i}$ is the attention weight at time t and h_i is the hidden state. The distraction was also placed immediately on the attention weight vectors. To do this, the previous attention weights were saved in a history attention weight vector and then added to the currently computed attention weights.

$$a'_{t,i} = \upsilon_a^T \tanh\left(W_a s'_t + U_a h_i - b_a \sum_{j=1}^{t-1} \alpha_{j,i}\right) \tag{21}$$

And,

$$\alpha_{t,i} = \frac{\exp(\alpha'_{t,i})}{\sum_{j=1}^{T_x} \exp\left(\alpha'_{t,j}\right)} \tag{21}$$

Where, υ_a, W_a, U_a and, b_a are the weight matrices.

2. DEVELOPMENT

2.1. PRESENTATION

We covered several deep learning technics for our Summarization Task. We will list all of our models and we will present what is the base code and our contribution on each of them.

- T5 Transformer: We based our code on an implementation that has as primary goal to call the pretrained model for various tasks. Our contribution is as follow: we created the data pipeline that will preprocess the data, put the data in shape ready to be processed by the transformer. We created the evaluation pipeline as well to evaluate our model on our particular dataset and results.

- T5 Transformer Finetuned: We based our code on an implementation that finetune the T5 transformer for a translation task. Our contribution aimed to change the finetuning for our task. The data preprocessing and the evaluation metric has been added as well.

- Seq2Seq Model: I followed a tutorial to understand correctly all the principles that lead to the creation of such basic deep learning model and to understand better the transformer architecture.

- Transformer based architecture model. We coded this transformer using all the knowledge we gathered during all our study. We will pass by a precomputed embedding as we are not well geared with very strong computational power and rams. Our model is very basic and serve the only purpose to learn from it.

In the next section I will present you more in depth the T5 transformer, from his history to his utilization for our task. We will :

- Introduce the t5 transformer, the idea behind it, the support and the pre-training the T5 transformer passed through.
- The finetuning concept and its implementation
- The dataset we selected
- The experimental framework we chose

2.2. T5 TRANSFORMER

2.2.1. INTRODUCTION

It has recently become more usual to pre-train the entire model on a data-rich assignment. This pre-training should ideally cause the model to gain general-purpose talents and knowledge that can later be transferred to downstream activities. Modern transfer learning techniques in NLP, on the other hand, frequently pre-train utilizing unsupervised learning on unlabeled data. This method has lately been utilized to get cutting-edge results in several of the most common NLP benchmarks (Devlin et al., 2018 [15]). Aside from its empirical strength, unsupervised pre-training for NLP is particularly appealing since unlabeled text data is abundant on the Internet—for example, the Common Crawl project2 generates approximately 20TB of text data retrieved from web pages each month. This is a perfect fit for neural networks, which have been demonstrated to have amazing scalability, i.e., it is frequently possible to gain greater performance simply by training a larger model on a larger data set (Hestness et al., 2017 [21]). Because of the quick rate of growth and diversity of techniques in this relatively young subject,

it can be difficult to evaluate different algorithms, separate out the impact of new contributions, and understand the space of existing transfer learning approaches.

The T5 transformer's main principle is to consider any text processing problem as a "text-to-text" problem, that is, to take text as input and produce new text as output. Importantly, the text-to-text framework enables us to apply the same model, objective, training approach, and decoding process to any task we can think of.

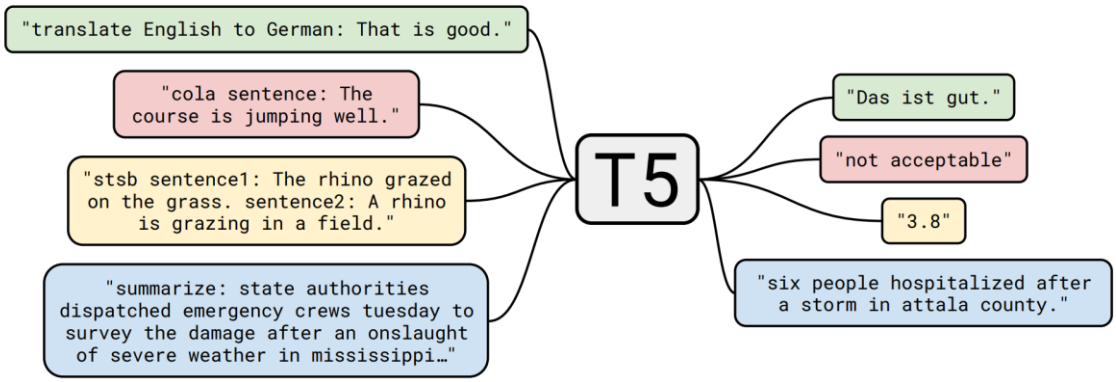

Figure 17: A diagram of the text-to-text framework.

(Image taken from: https://ai.googleblog.com/2020/02/exploring-transfer-learning-with-t5.html)

To do all tasks that we could wish, the T5 transformer team built the “Colossal Clean Crawled Corpus” (C4), a data collection consisting of hundreds of gigabytes of clean English text scraped from the web. Recognizing that the primary benefit of transfer learning is the ability to use pre-trained models in data-scarce scenarios, they provide their code, data sets, and pre-trained models available.

2.2.2. MODEL

In this section, I will present the initial transformer built by Vaswani et al. (2017) [3], from which the Google Brain team began to construct the T5 Transformer, and then we will present the revisions they did to end up with the T5 as we use it in our study.

So, in the next section, we will present the two primary elements of a transformer, the Encoder and the Decoder.

ENCODER

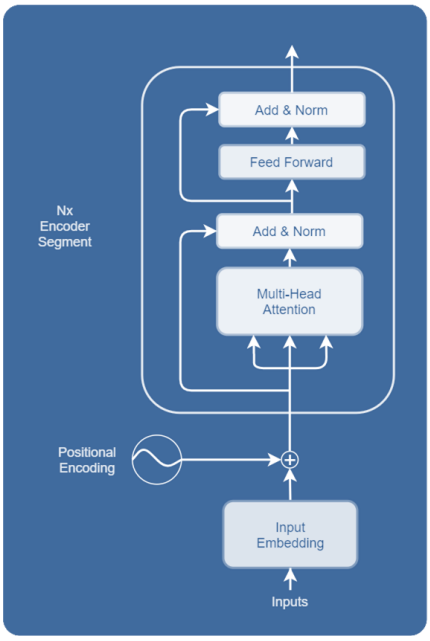

Figure 18: Architecture of the Encoder part.

When stacking encoders, the output of each encoder is utilized as the input for the next encoder, resulting in an increasingly abstract encoding. While stacking encoders can significantly enhance model performance through generalization, they are also computationally demanding. Vaswani et al. (2017) [3] chose N=6 and hence used six encoders layered on top of one another.

Each encoder segment is built from the following components:

- A **multi-head attention block**. This block enables us to execute self-attention over each sequence (i.e., for each phrase that we feed the model, identify on a per-token (per-word) basis which other tokens (words) from the phrase are important to that token; therefore, where to focus to while reading that token/word).
- A **feed-forward block**. Following the generation of attention for each token (word), we must construct a dmodel-dimensional, and hence 512-dimensional, vector that encodes the token. This is the responsibility of the feed forward block.
- **Residual connections**. A residual connection is one that does not traverse a complicated block. There are two residual connections seen here: one from the input to the first Add & Norm block, and another from there to the second block. Residual connections allow models to optimize more effectively since gradients may flow freely from the end of the model to the beginning.
- **Add & Norm blocks**. The result is then blended with the residual (by way of adding) from either the Multi-head attention block or the feed forward block, which results in a normalization of the layer.

DECODER

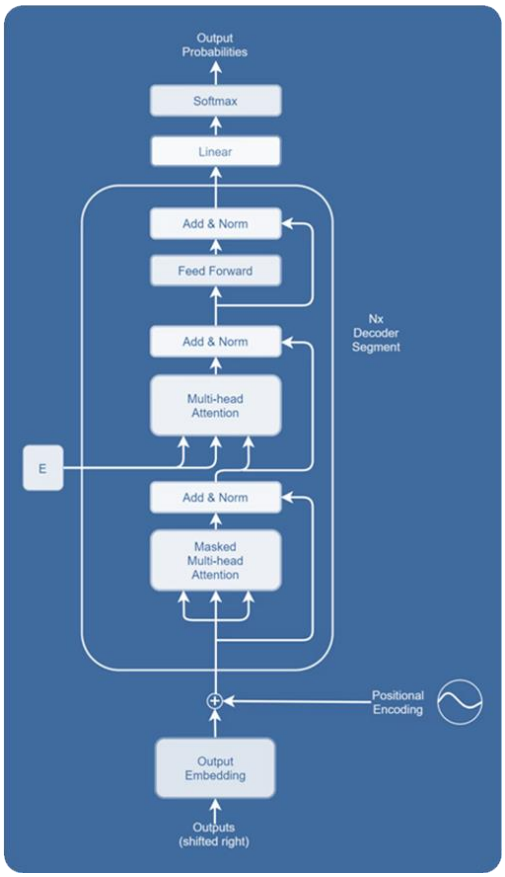

Figure 18: Architecture of the Decoder part.

Let's now take a look at the **decoder segment**. The intermediate, high-dimensional representation into predictions for output tokens is accountable for this part of a transformer. It appears visually like this. The segment decoder consists of several different components, and the segment decoder is repeated 6 times.

- **Output Embeddings**, which, like the embeddings used for the inputs, converts tokenized outputs into vector format. There is simply one variation here: the outputs are moved to one place. This, in conjunction with the masked multi-head attention segment, ensures that predictions for any position can only be based on known outputs at positions less than that input (Vaswasni et al., 2017) [3]. In other words, forecasts are certain to be based solely on the past and not on the future.
- **Positional Encodings**, which, like the input positional encodings, modifies the vector outputs of the embedding layer somewhat, adding positional information to these vectors.
- The **actual decoder segment**, which is composed of the following sub segments:
    - The **masked multi-head attention segment**, it performs multi-head self-attention on the outputs, but in a veiled manner, such that positions rely solely on the past
    - The **multi-head attention segment**, it uses multi-head self-attention on a mix of (*encoded*) inputs and outputs, allowing the model to learn to link encoded inputs with desired outputs
    - The **feed forward segment**, which processes each token individually.
- Finally, there is a linear layer that creates logits and a SoftMax layer that provides pseudoprobabilities (here is a tutorial to understand logits and pseudoprobabilities [22]). By calculating the argmax value of this prediction, we know which token should be taken and added to the tokens already predicted.

The Transformer was initially shown to be effective for machine translation, but it has subsequently been used in a wide variety of NLP settings (Radford et al., 2018 [23]; Devlin et al., 2018 [24] ; McCann et al., 2018 [25]).

To begin, an input sequence of tokens is mapped to an embedding sequence, which is then sent into the encoder. The encoder is made up of a stack of "blocks," each of which has two subcomponents: a self-attention layer and a tiny feed-forward network. Each subcomponent's input is subjected to layer normalization [26]. They employ a streamlined version of layer normalization in which activations are merely rescaled and no additive bias is introduced. A residual skip connection [27] adds each subcomponent's input to its output after layer normalization. Dropout [28] is used in the feed-forward network, on the skip connection, on the attention weights, and at the stack's input and output. The construction of the decoder is identical to that of the encoder, except that it adds a conventional attention mechanism after each self-attention layer that pays to the encoder's output.

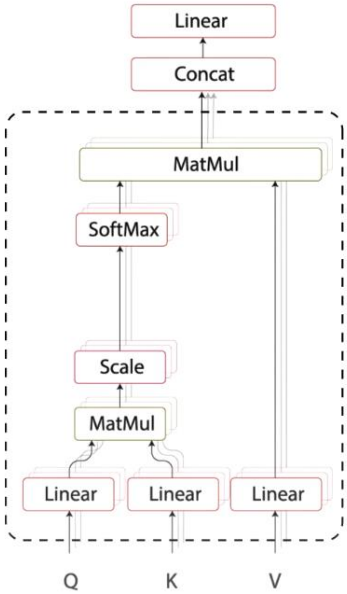

Figure 19: Self-attention mechanism

The decoder's self-attention mechanism also employs a type of autoregressive or causal self-attention, which only allows the model to attend to previous outputs. The final decoder block's output is fed into a dense layer with a SoftMax output, the weights of which are shared with the input embedding matrix. All attention processes in the Transformer are divided into independent "heads," the outputs of which are concatenated before further processing.

Because self-attention is an operation on sets and is order-independent, it is usual to supply an explicit position signal to the Transformer. A fixed number of embeddings are typically learnt, each corresponding to a range of potential key-query offsets. For this model, they utilize 32 embeddings with ranges that increase in size logarithmically up to an offset of 128 after which we give all relative locations to the same embedding. It should be noted that a particular layer is insensitive to relative position beyond 128 tokens, but future levels might acquire sensitivity to bigger offsets by merging local information from prior levels. To conclude, the model is nearly identical to the original Transformer provided by Vaswani et al. (2017) [3], with the exception of eliminating the Layer Norm bias, relocating the layer normalization outside the residual route, and employing a new position embedding strategy.

2.2.3. THE CRAWLED CORPUS

Much of the previous work on transfer learning for NLP makes use of large unlabeled data sets for unsupervised learning.

Common Crawl has already been utilized as a text data source for NLP, such as to train an n-gram language model [29] and to mine parallel texts for machine translation [30]. Common Crawl is a freely

accessible online archive that delivers "web extracted text" by extracting markup and other non-text information from scraped HTML files. Each month, this procedure generates around 20TB of scraped text data. Regrettably, the vast bulk of the generated content is not natural English. Instead, it's mostly made up of nonsense or boilerplate material, such as menus, error messages, or duplicate text.

Furthermore, most of the scraped text contains information that is unlikely to be useful for any of the jobs we are considering (offensive language, placeholder text, source code, etc.). To overcome these difficulties, they utilized the following algorithms to clean up the web retrieved text from Common Crawl:

- They only kept lines that concluded with a punctuation mark (i.e., a period, exclamation mark, question mark, or end quotation mark).
- They deleted any page with less than 5 sentences and kept only lines with at least 3 words.
- They eliminated any page containing any of the words on the "List of Dirty, Naughty, Obscene, or Otherwise Bad Words."
- Because several of the scraped websites featured warnings that JavaScript should be enabled, we deleted all lines that contained the term JavaScript.
- Some pages had placeholder “lorem ipsum” content; we eliminated any page that had the word “lorem ipsum”.
- Inadvertently, several pages included code. Because the curly bracket “” exists in numerous computer languages (including JavaScript, which is extensively used on the web), but not in natural text, we eliminated any websites that had one.
- In order to deduplicate the data set, they deleted all except one three-sentence span that appeared more than once in the data set.

Furthermore, because the majority of our downstream tasks are focused on English-language material, they utilized langdetect [31] to filter out any pages that were not categorized as English with a probability of at least 0.99. To create our base data set, we obtained the site extracted text from April 2019 and applied the aforementioned filtering. This results in a text collection that is not only orders of magnitude larger than typical pre-training data sets (around 750 GB), but also reasonably clean and natural English text. They dubbed this data collection the "Colossal Clean Crawled Corpus" (or C4 for short) and shared it.

2.2.4. INPUT AND OUTPUT FORMAT

To train a single model on the wide range of tasks described above, we convert all of the tasks under consideration into a “text-to-text” format—that is, a task in which the model is fed some text for context or conditioning and then asked to produce some output text. This framework establishes a constant training goal for both pre-training and fine-tuning. Before feeding the original input sequence to the model, we must add a task-specific (text) prefix to define which task the model should do.

For example, for our specific task of summarization:

Original input: I drink a soda on the beach

Processed input: summarize: I drink a soda on the beach

2.3. METRIC SELECTED

We tried several metrics as

- Levenshtein distance is the minimum number of single-character edits between two words
- The Lempel–Ziv–Markov chain algorithm (LZMA) is an algorithm used to perform lossless data compression. We compare the two compression with a simpler distance measurement
- Jaccard Distance is a measure of how dissimilar two sets are. The lower the distance, the more similar the two strings.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad (22)$$

- Mover's Distance (WMD), suggests that distances and between embedded word vectors are to some degree semantically meaningful.
- Rouge Score as it is widely used for text summarization evaluation. We will use ROUGE-N. It measures unigram, bigram, trigram and higher order n-gram overlap

We opted to keep WMD since it allows us to compare the meaning of the obtained summary to the one provided by the authors. We employ a word embedding technique based on word2vec and the Google News dataset (which contains over 100 billion words). Then, using the word-mover-distance python package, we use the Word mover Distance. WMD compares the word embeddings of two texts to determine the shortest distance that the words in one text must "travel" in semantic space to reach the words in the other text.

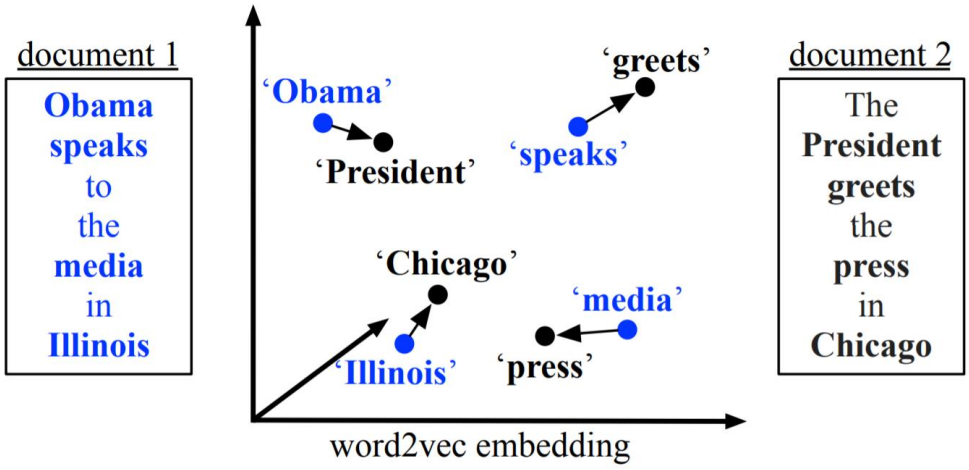

Figure 20. An illustration of the word mover's distance. All non-stop words (bold) of both documents are embedded into a word2vec space. The distance between the two documents is the minimum cumulative distance that all words in document 1 need to travel to exactly match document 2.

(Image taken from: [32])

2.3.1. WORD2VEC EMBEDDING

Word2vec, a revolutionary word-embedding approach, was recently introduced by Mikolov et al. (2013a; b) [32]. A two-layer neural network language model is used by their model to learn a vector representation for each word. To anticipate adjacent words, they suggest a neural network design (the skip-gram model) including an input layer, a projection layer, and an output layer. Each word vector is trained to maximize the log probability of neighboring words in a corpus, i.e., given a sequence of words $w_1, .., w_T$.

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{j \in nb(t)} \log p(w_j|w_t) \quad (23)$$

where $nb(t)$ is the set of neighboring words of word w_t and $p(w_j|w_t)$ is the hierarchical softmax of the associated word vectors v_{w_j} and v_{w_t} Due to its surprisingly simple architecture and the use of the hierarchical softmax, the skip-gram model can be trained on a single machine on billions of words per hour using a conventional desktop computer. The ability to train on very large data sets allows the model

to learn complex word relationships such as $\text{vec(Japan)} - \text{vec(sushi)} + \text{vec(Germany)} \approx \text{vec(bratwurst)}$and $\text{vec(Einstein)} - \text{vec(scientist)} + \text{vec(Picasso)} \approx \text{vec(painter)}$ Learning the word embedding is entirely unsupervised and it can be computed on the text corpus of interest or be pre-computed in advance.

Word2Vec can capture numerous degrees of similarity between words, allowing semantic and syntactic patterns to be replicated using vector arithmetic. Patterns like "Man is to Woman as Brother is to Sister" can be generated by performing algebraic operations on the vector representations of these words in such a way that the vector representation of "Brother" - "Man" + "Woman" produces a result that is closest to the vector representation of "Sister" in the model. Such links can be constructed for a variety of semantic relations (for example, Country—Capital) as well as syntactic relations (for example, present tense—past tense).

Assume we are provided with a word2vec embedding matrix $X \in R^{d*n}$ for a finite size vocabulary of n words. The $i\ th$ column, $xi \in R^d$, represents the embedding of the i th word in d-dimensional space. We assume text documents are represented as normalized bag-of-words (nBOW) vectors, $d \in R^n$. To be precise, if word i appears c_i times in the document, we denote $d_i \frac{c_i}{\sum_{j=1}^{n} c_j}$. Because most words will not exist in any given manuscript, a nBOW vector d is intrinsically quite sparse. (We eliminate stop words, which are often category agnostic.)

nBOW representation. The vector d can be thought of as a point on the n-1 dimensional simplex of word distributions. Two texts with distinct words will be found in various parts of this simplex. These documents may, however, be semantically related. Consider the previous example of two similar, but word-for-word distinct lines in one document: *"Obama speaks to the media in Illinois"* and *"The President greets the press in Chicago."* After the stop-word is removed, the two associated nBOW vectors d and d' have no common non-zero dimensions and hence have close to maximum simplex distance, despite the fact that their real distance is minimal.

Word travel cost. The semantic similarity between individual word pairs (e.g., *President and Obama*) will be incorporated into the document distance measure. The Euclidean distance between words in the *word2vec* embedding space provides one such measure of dissimilarity. More precisely, the distance between word i and word j becomes $c(i,j) = ||x_i - x_j||_2$. To prevent confusion with word and document distances, we shall refer to $c(i,j)$ as the cost of "traveling" from one word to another.

Document distance. The "travel cost" between two words is a natural building block to create a distance between two *documents*. Let d and d' be the nBOW representation of two text documents in the $(n-1)$ −simplex. First, we allow each word i in d to be transformed into any word in d 0 in total or in parts. Let $T \in R^{n*n}$ be a (sparse) flow matrix where $T_{ij} \geq 0$ denotes how much of word i in d travels to word j in d'. To transform d entirely into d' we ensure that the entire outgoing flow from word i equals d_i, i.e., $\sum_j T_{ij} = d_i$. Further, the amount of incoming flow to word j must match d'_j, i.e., $\sum_j T_{ij} = d'_j$. Finally, we can define the distance between the two documents as the minimum (weighted) cumulative cost required to move all words from d to d', i.e., $\sum_{i,j} T_{ij}\, c(i,j)$

2.3.2. ROUGE SCORE

We opted as well to make use of the rogue score.

ROUGE is an acronym that stands for Recall-Oriented Understudy for Gisting Evaluation. It is essentially a set of metrics for evaluating automatic text summarization and machine translation. It compares an automatically generated summary or translation to a set of reference summaries (typically human produced).

Precision and Recall in the ROUGE Context

Simply expressed, recall in the context of ROUGE refers to how much of the reference summary the system summary recovers or captures. If we only consider the individual words, it can be calculated as follows:

$$\frac{number\_of\_overlapping\_words}{total\_words\_in\_reference\_summary}$$

A machine generated summary (system summary) can be extremely extensive if it includes all of the terms in the reference summary. However, many of the words in the system overview may be unnecessary, making the summary overly verbose. This is where dexterity is required. In terms of precision, you're essentially measuring how much of the system summary was actually useful or necessary. Precision is expressed as:

$$\frac{number\_of\_overlapping\_words}{total\_words\_in\_system\_summary}$$

When attempting to develop brief summaries, precision becomes extremely important. As a result, it is always advisable to compute both the Precision and the Recall before reporting the F-Measure. If your summaries are compelled to be succinct due to constraints, you could try using only the Recall because precision is less important in this case.

The reason for using ROUGE-1 instead of or in addition to ROUGE-2 (or any finer granularity ROUGE measures) is to demonstrate the fluency of the summaries or translation. The idea is that if you closely replicate the word orderings of the reference summary, your summary will be more fluent.

For more in-depth information about these evaluation metrics, you can refer to Lin's paper [33].

We will make use of the Rouge-1 as we removed stop word for the evaluation.

2.4. FINETUNING THE MODEL

Finetuning is taking the weights of a trained neural network and using them as the basis for training a new model on data from the same domain (often e.g., images). It is employed to:

1. speed up the training
2. overcome small dataset size

There are several approaches, including training the entire initialization network or "freezing" some of the pre-trained weights (usually whole layers).

2.4.1. DATASET SELECTED

This is the dataset for the TL; DR challenge, which contains Reddit postings suited for abstractive summarization using deep learning. The file format is a json file, with each line containing a JSON object representing a post. Each post's schema is shown below:

- author: string (nullable = true)
- body: string (nullable = true)
- normalizedBody: string (nullable = true)
- content: string (nullable = true)
- content_len: long (nullable = true)
- summary: string (nullable = true)
- summary_len: long (nullable = true)
- id: string (nullable = true)
- subreddit: string (nullable = true)
- subreddit_id: string (nullable = true)
- title: string (nullable = true)

The dataset includes 3,084,410 postings with an average length of 211 words for the content and 25 words for the summary.

2.4.2. ACTUAL FINETUNING

PRE-PROCESSING

We encounter a computational power issue because NLP tasks are quite expensive. We needed less than 12 GB of RAM and 100 GB of hard drive space to operate with Google Collaboratory. This forces us to reduce the number of samples provided with the dataset by a small margin.

To begin, we remove non-essential columns such as ‘author,' ‘body,' ‘id,' ‘normalizedBody,' and ‘subreddit id.'

This left us with only three columns: ‘content,' ‘subreddit,' and ‘summary.' After that, there were still 3,084,410 lines in the dataframe, which was too much for our session, so we chose to reduce it to 300 k.

Table 1: Visualization of the dataset not totally cleaned.

	content	subreddit	summary
2	b'very late to the party, but this is a good o...	b'AskReddit'	b'gramps lost his teeth at the beach and I fou...

From here we had to remove the b’ in the beginning.

After further work we understood that 300k was still too much for the fine tuning and the limitation of 12 hours we have on Google Collaboratory. We finally finish on 100 samples.

To choose these 100 samples we decided to take a closer look at the ‘subreddit’ columns as they give us practically the domain of the sample. We removed all the subreddit close to videos games and not appropriate sections. Some summaries given by the authors were not relevant to the content, that means that some people wanted to ‘troll’ the initiative. To get rid of these corrupted sample and all summaries that are not alphanumeric, we created a little data pipeline to choose whether we get rid of the sample or no.

To achieve this goal, we:

- tokenize the words with any given tokenizing method.
- Remove the stop words with the nltk stop words library
- Count the number of same words in content and summaries
- Threshold all the summaries that don’t contains minimum 5 words that appears in the content.

Then we randomly selected a sample from this ‘cleaned’ dataset.

At this stage the dataset looks like that:

Table 2: Visualization of the dataset cleaned

	content	subreddit	summary
0	very late to the party, but this is a good one...	AskReddit	gramps lost his teeth at the beach and I found...
1	Almost two weeks ago now, I lost my job, which...	trees	Lost my job, friend now coming by twice a day ...
5	Most major governments actions are to insure t...	explainlikeimfive	The only way you can think Snowden is a traito...

To Finetune our model we had to follow a certain pattern:

We will be working with the data and preparing it for fine tuning purposes.

- A new string is added to the main article column summarize: prior to the actual article. This is done because T5 had similar formatting for the summarization dataset.

The final Dataframe need to be something like this:

text	ctext
summary-1	summarize: article 1
summary-2	summarize: article 2
summary-3	summarize: article 3

CREATION OF DATASET AND DATALOADER

- The revised dataframe is divided into an 80-20 test-validation ratio.
- The CustomerDataset class is supplied both data-frames for tokenization of the new articles and their summaries.
- The tokenization is accomplished by utilizing the length parameters supplied to the class.
- To develop train and validation data loaders, train and validation parameters are defined and provided to the pytorch Dataloader construct.
- These data loads will be sent to train () and validate () for training and validation action, respectively.
- The console displays the shape of the datasets.

NEURAL NETWORK AND OPTIMIZER

At this stage, we define the model and optimizer that will be used for training and updating the network's weights.

• To define our model, we utilize the T5ForConditionalGeneration.from pretrained("t5-base") command. T5ForConditionalGeneration extends our T5 model with a Language Model head. The Language Model head enables us to generate text based on the T5 model's training.

• For our project, we are utilizing the Adam optimizer.

TRAINING MODEL AND LOGGING TO WANDB

- We now log all of the metrics in the WandB project that we created earlier.
- Then we call the train () with all of the relevant parameters.

VALIDATION AND GENERATION OF SUMMARY

- Following the completion of the training, the validation stage is initiated.
- The model weights are not changed, as specified in the validation function. Based on the article text, we apply the fine-tuned algorithm to generate new summaries.
- After every 100th step, an output is printed on the console with a tally of how many steps are completed.
- The original and created summaries are combined into a list and returned to the main function.
- Both lists are utilized to build the final dataframe, which has two columns**: Generated Summary** and **Actual Summary**.
- To be analyzed, the dataframe is saved as a csv file.

2.5. OWN CREATED TRANSFORMER MODEL

2.5.1. FIRST SEQ2SEQ BASIC MODEL:

We can create a Seq2Seq model for any situation that contains sequential data.

The input for Neural Machine Translation is a text in one language, and the output is also a text in another language:

I love playing sports → Me encanta hacer deporte

Our goal is to develop a text summarizer that takes a long string of words (from a text body) and returns a short summary (which is a sequence as well). As a result, it is possible to model it as a Many-to-Many Seq2Seq problem.

We can set up the Encoder-Decoder in 2 phases:

- Training phase
- Inference phase

TRAINING PHASE

During the training phase, we will first configure the encoder and decoder. The model will then be trained to predict the target sequence, which will be offset by one timestep.

ENCODER

The entire input sequence is scanned by an Encoder Long Short-Term Memory Model (LSTM) with a word fed into the encoder at every time. The information will be processed every time, and the contextual information included in the input sequence will be captured.

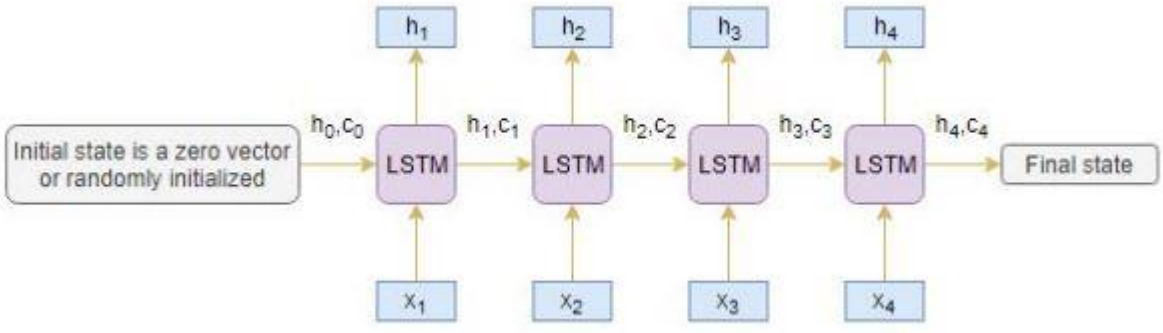

Figure 21: Usual Encoder pipeline for Seq2Seq models

(Image taken from: https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/)

The hidden state (h_i) and cell state (c_i) of the last time step are used to initialize the decoder.

DECODER

The decoder is an LSTM network like the encoder, which reads every word of a target sequence, predicting the same offset by one timestep. Throughout the past, the decoder is trained to predict the next word in the sequence.

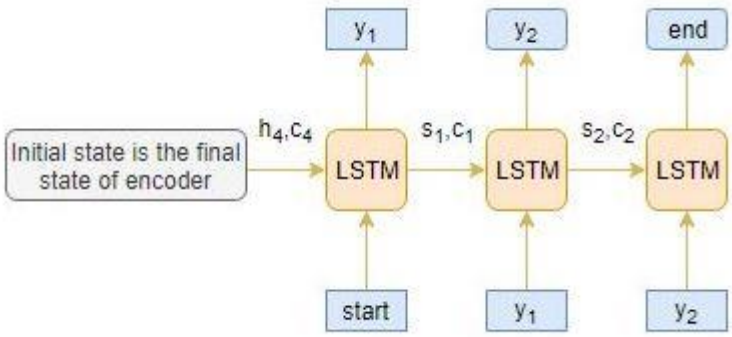

Figure 22: Usual Decoder pipeline for Seq2Seq models

(Image taken from: https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/)

The special tokens <start> and <end> are added to the target sequence before it is fed into the decoder. While decoding the test sequence, the target sequence is unknown. So, we begin predicting the target sequence by feeding the decoder the first word, which is invariably the <start> token. The <end> token denotes the end of the sentence.

INFERENCE PHASE

Following training, the model is tested on new source sequences with unknown target sequences. So, in order to decode a test sequence, we must first configure the inference architecture:

HOW DOES THE INFERENCE PROCESS WORK?

The following are the steps for decoding the test sequence:

1. Encode the whole input sequence and load the encoder's internal states into the decoder.
2. Provide the decoder with the <start> token as an input.
3. Run the decoder for one timestep given the internal states.
4. The likelihood of the next word will be the output. The word with the greatest chance of being chosen will be chosen.
5. In the next timestep, give the sampled word to the decoder as input and update the internal states with the current time step.
6. Repeat steps 3–5 until we generate an <end> token or reach the target sequence's maximum length.

LIMITATIONS OF THE ENCODER – DECODER ARCHITECTURE

This encoder-decoder architecture has significant drawbacks due to its simplicity.

• The encoder converts the entire input sequence into a fixed length vector, which the decoder then predicts. Because the decoder predicts based on the entire input sequence, this only works for short sequences, which is a problem for our dataset.

• This is where the problem with extended sequences originates. Long sequences are difficult to encode into a fixed length vector by the encoder.

To deal with our dataset, we introduced a layer of Global Attention mechanism to the Seq2seq model.

GLOBAL ATTENTION

The focus here is on all of the source positions. In other words, the attended context vector is derived from all of the encoder's hidden states:

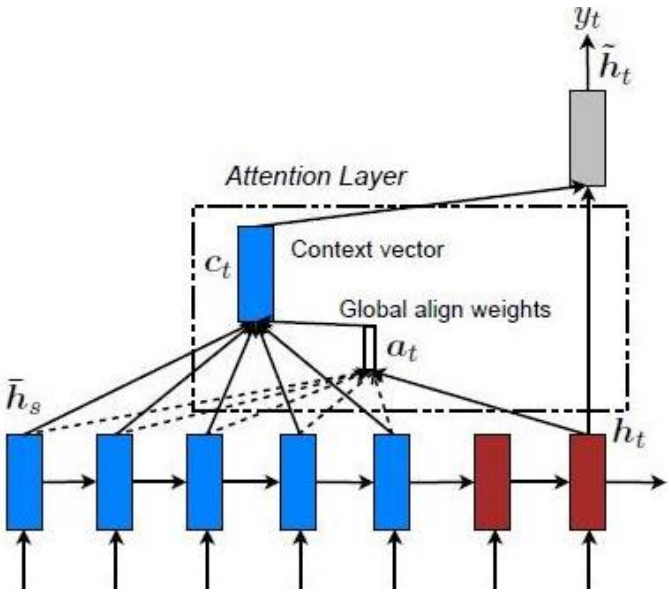

Figure 23: Global Attention layer developed for Seq2Seq models

(Image taken from: https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/)

Finally, a more coded view of our implementation can be summarized by this board:

```
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
input_1 (InputLayer)         [(None, 30)]              0
__________________________________________________________________________________________________
embedding (Embedding)        (None, 30, 100)           844000    input_1[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                  [(None, 30, 300), (N 481200    embedding[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)         [(None, None)]            0
__________________________________________________________________________________________________
lstm_1 (LSTM)                [(None, 30, 300), (N 721200    lstm[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)      (None, None, 100)         198900    input_2[0][0]
__________________________________________________________________________________________________
lstm_2 (LSTM)                [(None, 30, 300), (N 721200    lstm_1[0][0]
__________________________________________________________________________________________________
lstm_3 (LSTM)                [(None, None, 300),       481200    embedding_1[0][0]
                                                                 lstm_2[0][1]
                                                                 lstm_2[0][2]
__________________________________________________________________________________________________
attention_layer (AttentionLayer ((None, None, 300),    180300    lstm_2[0][0]
                                                                 lstm_3[0][0]
__________________________________________________________________________________________________
concat_layer (Concatenate)   (None, None, 600)         0         lstm_3[0][0]
                                                                 attention_layer[0][0]
__________________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 1989)     1195389   concat_layer[0][0]
==================================================================================================
Total params: 4,823,389
Trainable params: 4,823,389
Non-trainable params: 0
```

2.5.2. TRANSFORMER BASED ARCHITECTURE:

The general architecture and plan for our transformer will follow this graphic:

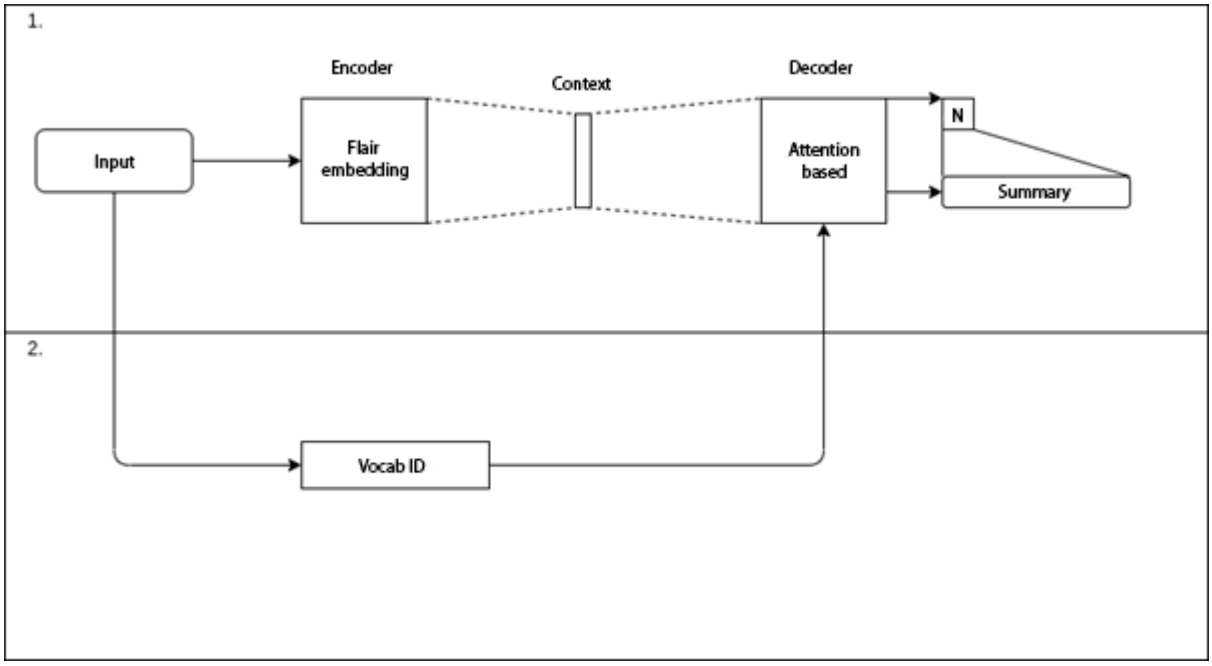

Figure 24: Global Architecture of our model

We followed two tutorials to achieve this goal:

https://towardsdatascience.com/how-to-code-the-transformer-in-pytorch-24db27c8f9ec

https://pytorch.org/tutorials/beginner/transformer_tutorial.html

Let's break all part of our code following the implementation chronology:

- Import, Installation of libraries, Data import
- Data preprocessing with creation of train set, validation set and test set
- Vocabulary creation using Sentence Piece (see this link)
- Embedding the data using Albert-Base embedding
- Defining the Pytorch dataset and Dataloader
- Defining the Decoder
- Train
- Evaluate

IMPORT, INSTALLATION OF LIBRARIES, DATA IMPORT

The imports are not very complexe as we are not trying to optimize a lot our model, we want from that code to learn from it so we want something very clear.

We import flair, sentencepiece, panda, numpy, torch and tensorflow.

We will work on our reddit300k dataset that contains approximately 160 k posts with their summary and they are cleaned before in our EDA notebook where we created all our dataset for our study.

DATA PREPROCESSING WITH CREATION OF TRAIN SET, VALIDATION SET AND TEST SET

60% of the dataset will be used for training and the rest for validation and test sets.

```
FULL Dataset: (164322, 2)
TRAIN Dataset: (98593, 2)
VALIDATION Dataset: (82161, 2)
TEST Dataset: (82161, 2)
```

We wants to make sure that the text data doesn't contains any uppercase character nor TensorFlow quotes that are demonstrated by this beginning b' . We previously did that job but manually when we created our dataset. This time we will just decode the NumPy array from TensorFlow in a UTF-8 convention.

VOCABULARY CREATION USING SENTENCE PIECE

The github from the creator of sentence piece explains it this way:

'SentencePiece is an unsupervised text tokenizer and detokenizer mainly for Neural Network-based text generation systems where the vocabulary size is predetermined prior to the neural model training. SentencePiece implements ***subword units*** *(e.g., byte-pair-encoding and unigram language model) with the extension of direct training from raw sentences. SentencePiece allows us to make a purely end-to-end system that does not depend on language-specific pre/postprocessing.'*

As you can see sentence piece permits a tokenization of a vocabulary regardless of its preprocessing it is the right candidate to construct a language model to improve our model performance.

We selected a vocabulary size of 30k.

EMBEDDING THE DATA USING ALBERT-BASE EMBEDDING

We need to instantiate our embedding base.

We chose the albert-base-V2 model as it is lighter than other pre-trained model but achieve the same level of performance than others big model like BERT and T5. This will allow us to have a lighter computation power need and to easily embed our sentences. And with his hidden state we want to perform a document pool embedding that consist of embedding not a single word but a complete document.

The simplest type of document embedding does a pooling operation over all word embeddings in a sentence to obtain an embedding for the whole sentence.

In our case a document is meant to be a sentence. So, we will pass all our dataset into this embedding to get number and to train our model with it.

DEFINING MODEL AND DATALOADER

This part defines the most important element of our work. It is the classes.

We have 3 class is our work:

- MyDataset
- PositionnalEncodding
- ContextDecoder
- TransformerModel

MYDATASET

This class tend to construct the tensor that will be used in the training. It basically got one static method that will cut the sentence into list of word. We will do that for the content and the summary.

POSITIONNALENCODDING

To understand a sentence, the model needs to know two things about each word: what it means and where does it stand in the sentence?

Because the embedding vector for each word will learn the meaning, we must now input something that informs the network about the word's position.

Vaswanii et al [3] answered this problem by using these functions to create a constant of position-specific values:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

This constant is a 2d matrix. *Pos* refers to the order in the sentence, and *i* refers to the position along the embedding vector dimension. Each value in the pos/i matrix is then worked out using the equations above.

CONTEXTDECODER

Here is a brief scheme of what we consider as a context in the transformer lexicon:

Word → Embedding → Positional Embedding → Final Vector, termed as Context.

This class is largely used by the next class as it is a part of the data processing pipeline. The aim of this class is to decode the context by creating a TransformerDecoderLayer

It is made up of self-attn, multi-head-attn and feedforward network. This standard decoder layer is based on the paper "Attention Is All You Need" [3].

We, here, create a one-layer transformer decoder layer. With the ReLu activation function, 2 head for the multi-head attention layer, and a dropout of 20%. We then define the forward function that will concentrate the context to a certain part of the input.

TRANSFORMERMODEL

The TransformerModel class concentrate all the model and make use of all previous classes.

In this class we define the predict-one function that will predict the summary.

Firstly, we instantiate all the previous class: PositionalEncoding, ContextDecoder.

We define two parts:

- The encoder: a TransformerEncoder layer with the same parameters as the one we presented in the ContextDecoder Class. We create an embedding instance as well in order to navigate in the embedding space using indices.
- The decoder: first we build a simple Linear decoder that we will improve with the help of the ContextDecoder layer that is a transformerDecoder layer.

We haven't constructed all the transformer explicitly, we used pytorch to create and instantiate encoders and decoders.

2.5.3. CONCLUSION

I used Pytorch to create a transformer that takes as input the precomputed embedding and the tokenized articles. I created one Part to handle the embedding, which should add up to a context every sentence, and another Part to handle the language's article input. I used an autoregressive technique to construct a Summary.

First, we created a method that uses flair embedding to obtain the Context Matrix, and then we used our decoder to obtain the summary.

Following that, we used the sentence fragment to obtain vocabulary in order to add additional information and so achieve a better result.

3. RESULTS

3.1. T5 TRANSFORMER PRE-TRAINED

The T5 transformer was tested on a Little subset of our original data of a span of 100 data simples.

As we explained the main metric selected will be Word moving distance and Human centered evaluation. We want the summaries to be short, very close to the human sones in meaning and conveying the clearest grammar in English.

For the pretrained Transformer there is some summaries:

Table 3: Comparison between Pretrained issued summaries with Human written summaries

Human Summary	T5_pretrained_Summary
Depression is nightmare, and it's hard to see someone suffer, but you can't deny yourself what you need.	it took me four years to get to this point, but I told her that I loved her and I just wanted her to be happy, whether that was with me or not. n I still struggle with depression, especially now. it's difficult for me to like myself when I don't see myself having any value to other people.
My first sushi experience in high school = no more animals at carnival.	the next day the principal announced no more live animals would be allowed at our carnivals any longer. the school carnival has a booth and someone said they would give us tickets for swallowing the goldfish they won from the biology students' booth.

3.2. T5 TRANSFORMER FINETUNED ON OUR DATA

The finetuning of the model has been done on a tiny subset of 100 samples cutter into two set as train, validation 80-20 percentage. The training loss indeed decrease as the training progress

Figure 25: Training Loss over time of our model

Table 4: Comparison between Finetuned issued summaries with Human written summaries

Human Summary	T5_finetuned_Summary
Told me he loved me when he was drunk, hasn't brought it up since...?	my boyfriend and I were at a party and became rather intoxicated, but neither of us has mentioned it since that night. Is it possible that we weren't being honest, or is he too nervous to say it sober? Do you tend to be more or less honest when you're drunk? Have you ever told him that you love each other? If so, do you have any
Learning to read is fucking hard if you're not a kid.	<extra_id_0> No one ever taught me how to read at a young age, and I still can not read quite as well as I would like, but at least now I can. Thank you parents, don't lie all responsibilities onto your teacher, I know you have jobs and bills and whatnot. This is kind of important. Learn simple things like math and reading for recreation. There is no window for learning,

3.3. MY OWN TRANSFORMER AND DEEP LEARNING MODEL

After 2 hour of training and a training loss that decrease, we saw that the Word moving distance is quite good.

Table 5: WMD for our own created transformer model

```
np.mean(wmd)

1.0105332106662197
```

But we clearly observe an overfitting on the value yield by our custom transformer because the summaries are not understandable:

Table 6: Comparison between our own created model issued summaries with Human written summaries

	Generated summary	Human Summary
0	all slam (10 lightly exhibition Valyria testi...	atheists should accept such condolences with ...
1	all slam (10 lightly exhibition Valyria testi...	find an ethnicity in your pool of percentages...
2	all slam (10 lightly exhibition Valyria testi...	Our petsitter did not communicate over the we...
3	all slam (10 lightly exhibition Valyria testi...	Learning to read is fucking hard if you're no...
4	all slam (10 lightly exhibition Valyria testi...	Started seeing this girl a few months ago and...
5	all slam (10 lightly exhibition Valyria testi...	I want to prove my pastor is an idiot who doe...
6	all slam (10 lightly exhibition Valyria testi...	I can't stop thinking about it, especially wh...
7	all slam (10 lightly exhibition Valyria testi...	Do you want to know how to beat the new raid ...
8	all slam (10 lightly exhibition Valyria testi...	I just want to hear your thoughts about curre...
9	all slam (10 lightly exhibition Valyria testi...	Friends constantly joke about massive dong in...
10	all slam (10 lightly exhibition Valyria testi...	Watching studio hosted tournaments without al...
11	all slam (10 lightly exhibition Valyria testi...	reddits commonly known but hard to use so the...
12	all slam (10 lightly exhibition Valyria testi...	After blasting me off my sparrow with his leg...
13	all slam (10 lightly exhibition Valyria testi...	My first sushi experience in high school <unk...

As we can see the model after training yield the same result which is not understandable.

We directly though about overfitting and the way to address this problem is to put more data into the training which is not easily done because of the limitations we are facing with google Collaboratory hardware repartition.

Unfortunately, we got the exact same problem with our seq2seq model as the data we used to train is only about 80 samples.

For the seq2seq model the output will looks like this:

```
Review: hi reddit please excuse drinking tonight ask reddit please let friends drink drive stopped friend met driving home tonight people playing
Original summary: do not your and in the and all of you do the
Predicted summary:  and and and and and and the the to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to

Review: actual joke around something made like even feel like complete harder harder think point guys joke live kid parents played joke think woul
Original summary: you do to and at you are an
Predicted summary:  and and and and and and the the to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to

Review: first time need income tax girlfriend income assets may indeed fafsa life may indeed go likely income whatever funds dependent taxes deper
Original summary: are not going on but is of and
Predicted summary:  and and and and and and the the to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to

Review: want nsa public place damn right stupid reddit mass nsa act know anyone support private property thing use video even breaking law along
Original summary: and we they if you your you are your to to with or
Predicted summary:  and and and and and and the the to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to

Review: hi everyone recently dating advice gave stuff read like hear area little month ago first year school struggling bit settle really know any
Original summary: the or
Predicted summary:  and and and and and and the the to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to to
```

We think that the problem is the same as the one before.

To address this problem, we can use different methods as

- Decreasing the learning rate
- Data augmentation
- Epoch augmentation
- Augmentation of the drop-out

We doubled the size of our data and multiplied by 5 the number of epochs. We passed the drop out % from 0.1 to 0.3.

But the result was different as the words yielded were different but the problem was the same as the model yield the same output for any input.

We think that we need to go deeper into these four methods but we are limited by the capability of google Collaboratory.

For the transformer we created we multiplied by 10 the numbers of epoch but the problem remains. I think that having way more data available for training could be a solution as the T5 transformer was pretrained on the C4 corpus which is approximately 750 Gb of texts. For example, our biggest dataset retrieved from the reddit challenge is 220 Mo (reddit300k_clean) And we used one sample of 100 data from this dataset so it even less than 10 Mo of data to train a transformer or a Seq2Seq model.

That’s why our two created models are not very good on our dataset.

To be sure that the problem is the one we explained we will train our algorithm on another dataset that has lesser long sentences and more clean texts. This dataset consists of reviews of fine foods from Amazon. The data spans a period of more than 10 years, including all ~500,000 reviews up to October 2012. These reviews include product and user information, ratings, plain text review, and summary. It also includes reviews from all other Amazon categories.

We’ll take a sample of 100,000 reviews to reduce the training time of our model. This dataset weight approximate 300 Mo but it is likely that we will not face the problem of overfitting because one sample is way less longer than the one, we had on Reddit dataset and the summaries often lies on two words.

The result is presented in the next figure. We can see the generated summaries and the target summary; we can see that the words are different but the meaning often time is the same. Secondly, we see that we don’t have the problem of overfitting so our hypothesis is validated.

Table 7: Test of our basic seq2seq model on another dataset to check if overfitting is still a problem.

	Generated summary	Human Summary
0	green tea	hour
1	yummy	would like to give it stars but
2	great coffee	delicious
3	salt salt	tastes ok packaging
4	great price	turkey jerky is great
...	...	...
95	great flavor	love the gum and the price
96	great deal	divine
97	great cereal	yes to smart bran
98	great flavor	good stuff
99	great tasting	yummy

The word moving distance metric yield this result for a test sample of 100:

Table 8: WMD for the Seq2Seq basic model.

```
np.mean(wmd)

1.5483897048105861
```

3.4. COMPARISON

Based on the len of the summaries we can see that the finetuned model achieve a longer summarization which is not very good as we expect to have as little as word in a summarization.

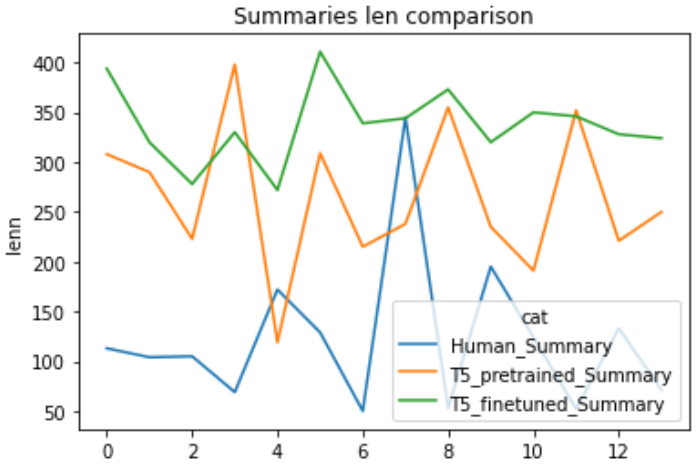

Figure 26: Summaries Len comparison

A contrary finetuning the model even on a very small training set permit us to improve a little bit the result on the metric we selected. It is surely due to the fact that finetuning the model on a particular dataset permit him to be accustomed to the vocabulary they typically use on the dataset and hence be better on this particular metric.

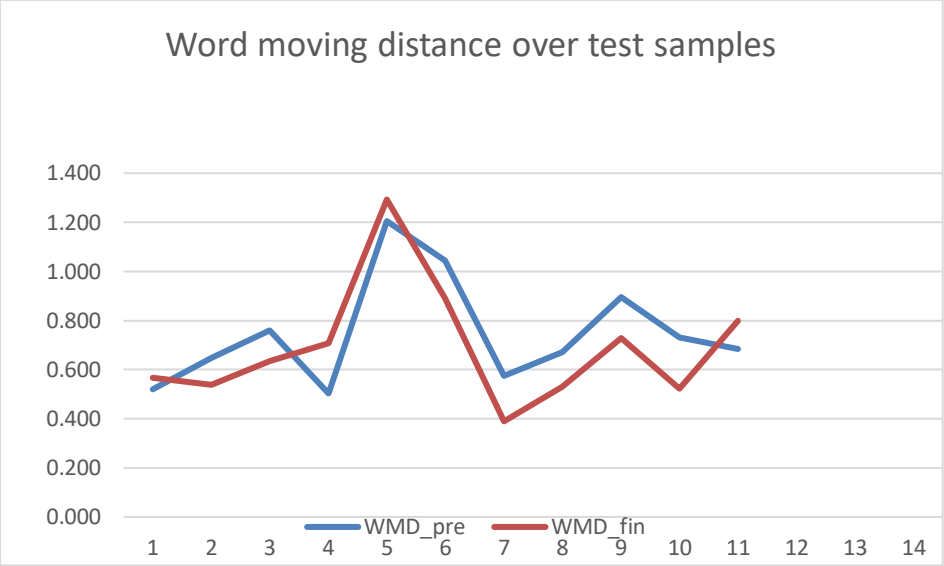

Figure 27: Word moving distance over test samples

Table 9: Result for WMD and Rouge score on Finetuned T5 transformer and pretrained T5 transformer

	Mean (words moving distance)	Rouge Score (F1 score)
Finetuned_T5 model	0.65	22.99
Pretrained_T5 model	0.71	18.82

4. CONCLUSION AND FUTURE LINES OF RESEARCH

4.1. CONCLUSION

Our work has been very informative and exhaustive for me to understand well the main force behind text summarization using deep learning. Our goals to review and learn from state-of-the-arts models has been widely completed as we used t5 transformer, made usage of albert base embedding basis, used a basic seq2seq model and finally finetune a pretrained model. We successfully been able to use already existing codes and modify them for our own usage. We were able to create our own transformer-based architecture and we were able to understand what didn't work for our results and test our hypothesis.

After all this work I feel very comfortable saying that a metric for NLP task is a very difficult thing to find correctly, I think that we should be very cautious about how we will measure any improvement In our results as a model that overfit will yield very interesting results for the metric we adopted.

4.2. FUTURE LINES OF RESEARCH

Firstly, to improve our work, I think that we can try to clean more our dataset and train our custom transformer on more data. We will be happy to overcome the problem of overfitting. It would be interesting as well to create application based on this summarization task. Document summarization is a very promising task as it can find precious usage for research, data collection and project preparation. Secondly, we can think about combining Automatic Speech Recognition with Text summarization to permits people to absorb large amount of spoken data into a well written summary. The results we observed during this study shows that we still have a great way to evolve but it is promising. We would love to finetune our model to yield specific type of summary as bullet-point summaries, paragraph summary or even for people who cannot read spoken summaries.

5. BIBLIOGRAPHY

[1] p. 1. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP).

[2] Minh-Thang Luong, Hieu Pham, Christopher D. Manning, Effective Approaches to Attention-based Neural Machine Translation, arXiv:1508.04025, arXiv:1508.04025.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, ‘‘Attention is all you need,’’ in Proc. Adv. Neural Inf. Process. Syst., NIPS, 2017, pp. 5998–6008. [Online]. Available: https://proceedings.neurips.cc/p.

[4] Transformer-XL: Attentive Language Models arXiv:1901.02860v3 [cs.LG] 2 Jun 2019,Zihang Dai∗12, Zhilin Yang∗12, Yiming Yang1 , Jaime Carbonell1 , Quoc V. Le2 , Ruslan Salakhutdinov1.

[5] "Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, Llion Jones; Character-Level Language Modeling with Deeper Self-Attention, arXiv:1808.04444".

[6] https://arxiv.org/abs/1901.02860.

[7] *Q. Chen, X. Zhu, Z. Ling, S. Wei, and H. Jiang, ‘‘Distraction-based neural networks for document summarization,’’ in Proc. 25th Int. Joint Conf. Artif. Intell., 2016. [Online]. Available: https://arxiv.org/abs/1610.08462.*

[8] *S. Chopra, M. Auli, and A. M. Rush, ‘‘Abstractive sentence summarization with attentive recurrent neural networks,’’ in Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Human Lang. Technol., 2016, pp. 93–98..*

[9] *K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, ‘‘Learning phrase representations using RNN encoder–decoder for statistical machine translation,’’ in Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP.*

[10] *D. Bahdanau, K. Cho, and Y. Bengio, ‘‘Neural machine translation by jointly learning to align and translate,’’ in Proc. 3rd Int. Conf. Learn. Represent., ICLR, 2015, pp. 1–15..*

[11] *S. Hochreiter and J. Schmidhuber, ‘‘Long short-term memory,’’ Neural Comput., vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/ neco.1997.9.8.1735.*

[12] *J. L. Elman, ‘‘Finding structure in time,’’ Cognit. Sci., vol. 14, no. 2, pp. 179–211, Mar. 1990, doi: 10.1016/0364-0213(90)90002-E.*

[13] *J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, ‘‘Empirical evaluation of gated recurrent neural networks on sequence modeling,’’ in Proc. NIPS Workshop Deep Learn., 2014..*

[14] *M. Schuster and K. K. Paliwal, ‘‘Bidirectional recurrent neural networks,’’ IEEE Trans. Signal Process., vol. 45, no. 11, pp. 2673–2681, 1997..*

[15] *J. Devlin, M. Chang, K. Lee, and K. Toutanova, ‘‘BERT: Pre-training of deep bidirectional transformers for language understanding,’’ in Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol., 2019, pp. 4171–4186..*

[16] *A. Radford, ‘‘Improving language understanding by generative pretraining,’’ OpenAI J., to be published..*

[17] *M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, ‘‘BART: Denoising sequence-tosequence pre-training for natural language generation, translation, and comprehension,’’ in Proc. 58th Annu. Meeting Assoc. Co.*

[18] *J. Gu, Z. Lu, H. Li, and V. O. K. Li, ''Incorporating copying mechanism in sequence-to-sequence learning,'' in Proc. 54th Annu. Meeting Assoc. Comput. Linguistics (Long Papers), vol. 1, 2016, pp. 1631–1640, doi: 10.18653/v1/p16-1154..*

[19] *A. See, P. J. Liu, and C. D. Manning, ''Get to the point: Summarization with pointer-generator networks,'' in Proc. 55th Annu. Meeting Assoc. Comput. Linguistics (Long Papers), vol. 1, 2017, pp. 1073–1083, doi: 10.18653/v1/P17-1099..*

[20] *Q. Chen, X. Zhu, Z. Ling, S. Wei, and H. Jiang, ''Distraction-based neural networks for document summarization,'' in Proc. 25th Int. Joint Conf. Artif. Intell., 2016. [Online]. Available: https://arxiv.org/abs/1610.08462.*

[21] *Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. arXiv preprint arXiv:1712.00409, 2017..*

[22] *https://www.machinecurve.com/index.php/2020/01/08/how-does-the-softmax-activation-function-work/#logits-layer-and-logits.* [Last accessed 21/06/2021]

[23] *Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language.*

[24] *Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pretraining of deep bidirectional transformers for language understanding. arXiv preprint.*

[25] *Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. The natural language decathlon: Multitask learning as question answering. arXiv preprint.*

[26] *Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016..*

[27] *Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, 2016..*

[28] *Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 2014..*

[29] *Christian Buck, Kenneth Heafield, and Bas Van Ooyen. N-gram counts and language models from the common crawl. In LREC, 2014..*

[30] *Jason R. Smith, Herve Saint-Amand, Magdalena Plamada, Philipp Koehn, Chris CallisonBurch, and Adam Lopez. Dirt cheap web-scale parallel text from the common crawl. In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics,.*

[31] *https://pypi.org/project/langdetect/.* [Last accessed 21/06/2021]

[32] *Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. In Proceedsings of Workshop at ICLR, 2013a..*

[33] P. Taylor, Text-to-Speech Synthesis, Cambridge University Press, 2009.

[34] Zen, H., Tokuda, K., & Black, A. W., "Statistical parametric speech synthesis," *Speech Communication, 51(11),* pp. 1039-1064, 2009.

[35] Pardo, J.M., Giménez de los Galanes, F.M., Vallejo, J.A., Berrojo, M.A., Montero, J.M., Enríquez, E., Romero, A., "Spanish text-to-speech, from prosody to acoustics," in *International Congress of Acoustics*, 1995.

ANEXO A: SUMMARIES SEQ2SEQ

Generated summary	Human Summary
green tea	hour
yummy	would like to give it stars but
great coffee	delicious
salt salt	tastes ok packaging
great price	turkey jerky is great
the best	my favorite salad dressing
not good	stale
great cat food	wonderful
my dogs love these	greenies buddy treat
great treats	dog loves them
great coffee	makes great cup of java
great training treat	love zuke mini naturals
delicious	protein bar
not decaf	not decaf
great cookies	cookie
great flavor	delicious but not the best
great coffee	hit or miss
great bread	really good gluten free bread
great snack	really good nice snack
horrible taste	please do not waste your money
my puppy loves these	healthy treats
good coffee	no flavor
rip off	water
yummy	new favorite
horrible taste	do not be by the price
good stuff	too expensive
great for	good value
good tea	caffeine is not
great product	excellent product for babies and toddler
dented cans	sometimes dented
great flavor	excellent love the blueberry pecan
great product	chew away
great plant	very nice office plant
my dog loves these	great dog treat
dogs love it	good for chewers
great tea	great price for great tea
my dog loves these	waste of money
delicious	fabulous product
great treat	dog lover
greenies	our dogs love greenies
good stuff	good stuff
not as advertised	wrong item
not bitter	disapointed
delicious	the yummy
great product	my dog loves it
great product	great to
great treats	buddy biscuits
great tea	green tea
the best	best taste
good	yummy tomatoes good packaging
great cookies	delicious
stale	baby loves it
too salty	no cheese flavor
great coffee	great morning coffee
great flavor	it made me fall
best coffee ever	single worst coffee ever
great snack	am addicted to these
great product	easy to use
great coffee	great coffee great price
refreshing	great taste all natural
good product	very good
great tea	decaffeinated french vanilla tea yummy
great taste	nice little pick me up
great product	excellent but
great coffee	wonderful coffee
good but not great	nice but pricey
great cookies	mouth watery cookies
great chips	pop chips are the best
hot cocoa	bold flavor
love these	perfect for in days
great	sun dried tomato bliss
great soup	soup chicken cheese
my dog loves these	strong smell and my puppy loves it
great snack	great purchase
great flavor	good wine
salt	over the top seasoning
great coffee	wolfgang puck coffee vanilla
not the best	edible have had much better
not bad	my favorite of the four tried
great snack	great snack
great price	love it
great for on the go	cannot get enough
great product	am an adult still love this
great tasting	you can eat ginger again
not for me	can for your
good product	wheat grass seeds
great product	fun like dust
great product	jumbo greenies good but very expensive
good but not great	very good but too pricey
great product	wonderful
just ok	better than regular
cats love it	need to know how many in case
great taste	great taste
great value	great tea
great taste	blue mayo
great flavor	love the gum and the price
great deal	divine
great cereal	yes to smart bran

ANEXO B: SUMMARIES COMPARISON

Human Summary	T5_pretrained_Summary	WMD_pre	T5_finetuned_Summary	WMD_fin
Physicists started using math as play blocks, and built some pretty sweet castles that can describe the universe,	math is very useful in solving problems, n that's pretty simple, but it works really for the simple stuff, it can describe how much pressure is in an oven as it heats up, or being able to determine the change in frequency of light from a distant star, that leap is pretty significant when you think about it,	0,518	people started using math that seemed to be describing simple things, and they started expanding it, It works really for the simple stuff, like finding how far you can walk in an hour, That leap is pretty significant when you think of it, but it's still huge and complicated, How is the crazy math still helping us solve problems? n Well, it started simple, People started building castles with	0,566
Depression is nightmare, and it's hard to see someone suffer, but you can't deny yourself what you need,	it took me four years to get to this point, but I told her that I loved her and I just wanted her to be happy, whether that was with me or not, n I still struggle with depression, especially now, it's difficult for me to like myself when I don't see myself having any value to other people,	0,649	n I still struggle with depression, especially now, It's difficult for me to like myself when I don't see myself having any value to others, but the important part is that it is my issue, not everyone elses, She left me and for the longest time I felt betrayed, like she had bailed on me in my time of need, I met up for	0,538
Told me he loved me when he was drunk, hasn't brought it up since,,,?	three days ago, my boyfriend and I were at a party and became intoxicated, he said that we loved each other, but neither of us has brought it up since that night, so I doubt it forgot that part, ours is his first serious relationship so i don't want to make him nervous and am going to wait for his to bring up, I'm just wondering if you all have any ideas as to why we might not be saying anything	0,759	my boyfriend and I were at a party and became rather intoxicated, but neither of us has mentioned it since that night, Is it possible that we weren't being honest, or is he too nervous to say it sober? Do you tend to be more or less honest when you're drunk? Have you ever told him that you love each other? If so, do you have any	0,634
speech is (or at least requires) money, If you don't regulate it, the people with the most money get to do the most talking, In e	n the problem is that speech is money, and a small number of people have most of the money, the current system puts the rich up on the stage with officials, gives them fliers on everybody's car, while those few people who care to speak up for everybody or anybody else have to try and talk over the megaphone,	0,503	Undue influence of corporations (for profit, or not for profit), Because corporations have access to resources greater than the average citizen does, Similarly, many non-profits, on the right and the left, are funded by a relatively small number of donors who give big, For an election to be sufficiently representative, you have to guard against both the richest among us, and corporations, flooding the nation	0,708
Saw cousin play Cod4 on his gaming pc, got hooked,	the first console I had was the Nintendo 64 and I loved it, as I got older I obtained more systems such as the PS2, Xbox, and Gamecube, I then moved on to maplestory and wasted my life on that for a number of years,	1,205	the first console I had was the Nintendo 64 and I loved it, As I got older I acquired more systems such as the PS2, Xbox, and Gamecube, At the same time I was playing my first PC game: Survival Project, He let me play and ever since then I've been a console gamer, Cod4 is one of the best games I ever played on the server I usually played	1,293
Learning to read is fucking hard if you're not a kid,	nNobody ever made me read books at a young age, or even how to read in general, the first word I ever read, it was near the end of second grade, about two years after most folks learn, and teacher asked me to write the word - "Baby?" "that's right!" 'nI still can not read quite as well as I would like, but the problem I had where I forget what I read is	1,043	<extra_id_0> No one ever taught me how to read at a young age, and I still can not read quite as well as I would like, but at least now I can, Thank you parents, don't lie all responsibilities onto your teacher, I know you have jobs and bills and whatnot, This is kind of important, Learn simple things like math and reading for recreation, There is no window for learning,	0,889
I can't stop thinking about it, especially when I'm writing a full length book about our nearly two decade friendship, ugh and I want him to read it when I'm done but he's not even talking to me,	the first weekend we met in person was like a dream but it really happened, n we've been through so so much and over came sixteen long hard years of long distance,,,and now he has commitment issues when distance is no longer an excuse,	0,574	n I don't want to think about it anymore, I wish I would kiss him again and never let him go and I'll love him like no one ever has loved him before, but he won'T let me forget it, He was the last standing friend who knew me almost two decades ago and still is there for me today, That's why I need to keep in touch with	0,389
Do you want to know how to beat the new raid as quickly as possible, or are you going to try and figure it out for yourself?	year 2 player has never had the fun (and frustration) of struggling through a brand new raid with unknown mechanics, guide, youtube videos, sherpas, etc, all of which was great! very helpful,	0,670	n Got to thinking though, will you be atttempting a brand new raid with your buddies totally blind, or are youtubing and reddit browsing and trying to beat it as quickly and easily as possible? All of which was great! Very helpful, Have you ever had the fun (and frustration) of beating it with unknown mechanics? Are you going to be doing all of the	0,530
reddits commonly known but hard to use so they don't	reddit is complex to use because it's complex, it doesn't use it because its complex and complex, a lot of people know about it but people around me know what it is but they know if it comes up they donÃ¢Â€Â™t have it - it makes it difficult to rely on it so it can be very expensive and expensive but it has the most expensive price and is not expensive,	0,894	ya alot of people don't know about it but most of the time when reddit comes up the people around me know what it is but they never use it because its complex to use and if it does come up its easy to understand but its not easy because it's complicated to find and it takes time to get it up and use so many times its difficult to explain it and	0,730
After blasting me off my sparrow with his legendary Raid sparrow, he tried it again only to glitch himself into the cab of a truck,,,	my friend received his legendary raid sparrow a couple weeks back, this happened while on the way to the Daily heroic, n (you don't see how it happened but you can hear the explosions in the background when it did happen)	0,731	after my friend received his legendary raid sparrow a couple weeks back, this happened while on the way to the Daily heroic, nEdit: To clear up any confusion by 'glitch' I mean he's stuck in the cab of the truck, No way in or out, Edit: This is not the same thing as the original story, This happened during the daily heroic and	0,522
My first sushi experience in high school = no more animals at carnival,	the next day the principal announced no more live animals would be allowed at our carnivals any longer, the school carnival has a booth and someone said they would give us tickets for swallowing the goldfish they won from the biology students' booth,	0,685	the next day the Principal announced no more live animals would be allowed at our carnivals any longer, We started a booth and someone said they would give us tickets for swallowing the goldfish they won from the biology students' booth, But we profited nicely, but we were disappointed when they announced we had to cancel,	0,799