Final Report

State Of The Art Text Classification Using Neural Architectures

SYS843 Neural Networks and Fuzzy Logic

Ecole de Technologie Supérieure, Montréal

Louis CARON

April 2020

Contents

1 Introduction

Computers are extremely fast to analyse and process standardized data. Although transforming real-world data was never easy, a lot of tools helped improving this task. While most of today's information is stored using computer technology, a lot of information needing processing still is under 'human' form. This information, written in *natural language*, can currently only be fully used and understood by humans. It includes mails, conversations, news, *etc.*..

Being able to fully process this information would mean a better integration of technologies into our daily lives, and bring many economical opportunities. From targeted marketing and virtual assistants to trading bots reading the news, *Natural Language Processing* is a dynamic domain between linguistic and data science studied by many economical and academical entities. The private sector is heavily investing in it, with the famous *GAFAM* spearheading most of the research and latest breakthroughs in this sector.

NLP is already being used on the market, with virtual assistants like *Siri* of *Apple* being able to understand a variety of sentences an to answer accordingly to questions.

In finance, trading bots are able to react to external events such as the recent attacks on Saudi oil facilities. A bot being able to read and process the news would generate billions, reacting faster than any human to such situations.

Text processing is the branch of NLP specialized in the treatment of textual data. It currently is one of the most researched topic in NLP and has many economical applications. Indeed, most of today's world information is stored as text on a digital media.

This includes general knowledge databases like Wikipedia, but also our mails, messages, social media accounts and more. Text processing is therefore very interesting economically, for targeted marketing for instance.

Text processing can also be used for many tasks such as the summarization of a corpus of texts, which would improve the productivity of many people around the world.

Text processing can itself be split in several sub-fields, such as text summarization, text generation, next sentence prediction or text classification.

In this project, we will only study text classification, as *NLP* and even text processing are too wide to be studied globally. Text classification is a subcategory of *NLP* aiming to process and understand text by classifying it among different categories. This can for example be used for chatbots or in finance as stated above.

1.1 Problem statement

Text processing aims to classify sentences or words by their meanings in order to draw information from them, this is called *semantic classification*. To reach the goal of classifying documents, the text must be transformed into data and vectorized before being classified. However, *natural languages* are way more complicated and less explicit than computer languages. Different words or sentences can have the same meaning, whereas small differences can lead to completely different meanings.

Natural language has a structure, and an infinity of potential sentences can be made. Moreover, some 'human' trait such as irony are really hard to grasp for a computer. The same goes for complicated text with double negatives, sentences making references to past discussion ("What did you think of that place?"), messages needing context, *etc...*

Overall, discriminating and classifying a text requires to grasp its meaning, and this meaning can only be understood by taking into account the relationships between words, their "context".

Despite the inherent complexity of the textual data, the recent progress of NLP, brought new methods to classify text. Those techniques all are neuronal approaches. This includes CNN, RNN and especially transformers [1] [2] [3] [9].

Transformers are neuronal units invented in 2017 for text translation tasks. They have since been applied to other NLP tasks including classification, and are considered to be a revolution in text processing.

Since their apparition, many models based on transformers have been engineered, such as BERT [9]. BERT was considered - and still is considered - as the sate of the art for many text processing tasks.

Unfortunately, BERT is also very heavy, which is why DistilBert [14] and ALBERT [15] were created. Those models are based on BERT, and they aim to conserve its performance while being as light as possible. To achieve this, they use different techniques of parameter reduction and pruning.

Given the importance of text classification, it is relevant to understand and compare the different methods being used nowadays to classify texts. This implies to study and compare the state of the art models BERT, DistilBert and ALBERT [9] [14] [15]. DistilBert and ALBERT are both based on BERT and their respective papers claim them to have great performance while being lighter than BERT.

It would therefore be interesting to study and characterize those state of the art models, and to see how they compare to each others. Indeed, this might give us an insight on the real necessity for huge models, which have become more and more common over the years.

1.2 Aim of the project

As said in the previous section, text classification has greatly improved thanks to the use of neural networks and mainly because of transformers [9].

At the moment, the state-of-the-art techniques for text classification are the most recent transformers architectures such as BERT [9], DistilBert [14] and ALBERT [15]. While they all use transformers, those models all have different architectures. Moreover, DistilBert and ALBERT make use of parameter reduction techniques and are designed to be lighter and faster than BERT.

In this project, we aim to answer to several questions: How does BERT perform on text classification tasks? How does it compare to DistilBert and ALBERT? Is the complexity of huge models really necessary for such tasks?

To answer those question, we plan to analyze BERT, DistilBert and ALBERT and to provide an exhaustive and impartial comparison of those models. The comparison will be focused on text classification tasks, and will include testing the accuracy of each model as well as its training and evaluation speed.

The results of the benchmark can be used by anyone wishing to implement a text classification model. It can also give a hint on how the models would behave on different tasks. Moreover, it can give critical information regarding the required complexity of a model for text classification: Is the complexity of BERT really necessary? are the parameter reducing techniques efficient?

In this report, we will start by giving an overview of text classification and of the literature regarding the aforementioned models : BERT, DistilBert and ALBERT.

Then, we will discuss the methodology of the project. The databases, metrics and protocol will be made explicit to ensure reproducibility and impartiality of the experiments.

We will then present the results of the experiments and discuss the results. We will also provide interpretation for the results. The results can be used by anyone wanting to use the studied models or to implement a new one.

Finally, we will conclude by giving the key points and takeaways of the project and by giving opinions about what to do next.

2 Overview of text classification

In this section, we will introduce and review the state of the art models used in this project. The models are BERT [9], DistilBert [14] and ALBERT [15]. We will start by giving a general overview of text classification from a historical perspective. Each model will then be reviewed and discussed.

2.1 Generic text classification

We will start by presenting text classification and some of the models historically used for this task.

Text classification is a broad generic term that refers to the classification of textual data into one or several categories. Said data can vary in length and type. For instance, classifying short sentences, news articles, or entire law documents containing thousands of words is considered text classification.

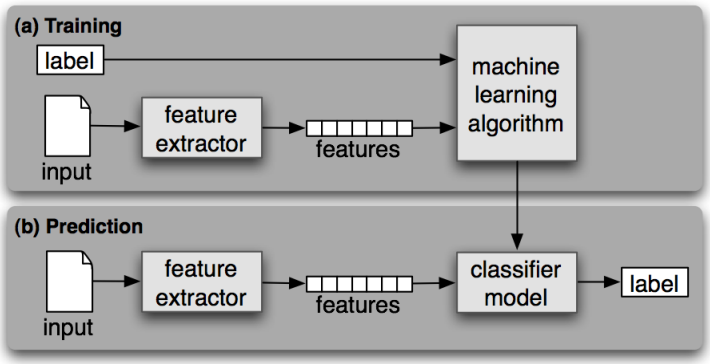

Figure 1: High level description of text classification (credits to nltk.org)

As shown in Fig 1, text classification is similar to any other type of classification. For any given input, a feature extractor gets the feature vector which is then fed to a machine learning algorithm (a neural network in our case) with its label during training. Once training is over, the network can be used to predict the label of an unknown sample.

Despite this, text classification also possess some noticeable particularities. Due to the nature of the data to classify, text classification faces several challenges compared to "regular" classification.

The first challenge faced by text classification systems resides in the use of data of various length. Texts and words can have different lengths and choosing the appropriate one for a model can prove to be difficult.

In almost all papers, the words have to be vectorized into a fixed-length vector [10] [4] [9]. The text itself must be represented as a sequence of vectors and must be of a fixed length, no matter the number of words in the sequence.

In order to fix the length of each text sample, we can define a maximum length and simply pad or

truncate all samples to this size. This method is used in most papers to accommodate text samples of different sizes.

To project words of different sizes into fix sized vectors, an embedding must be used. There are many different embedding techniques.

Some are quite simple, like WordPiece [5] which decomposes words to their simplest root and uses a dictionary of 30 000 words to generate a number for each word ("Eating is great" $\rightarrow$ " Eat + ##ing + is + great" $\rightarrow$ "123 + 456 + 789 + 321" for instance).

Others are more complex, like Word2Vec [4] or GloVe [11], which create a vector space in which the distance between each vector represents the semantic similarity between the words. Two words having similar meaning will have a short distance between their vectors if we use these embeddings.

The use of an embedding and the padding/truncating of a text allow to represent any given text into a fix-sized matrix or fix-sized sequence of vectors.

The second challenge with text classification as well as almost all NLP tasks is the complexity of human language. Complex traits like irony and sarcasm are poorly understood by machines.

The sentence "You truly are a genius" can be an example of sarcastic commentary made by people on a daily basis. Once again, the need for additional contextual information is high, as such sentence could be genuine or not depending on the contextual situation.

Although those very "human" traits are hard to deal with, neural approaches have been able to process them with some success. The field of sentiment analysis specialize in classifying complex texts involving human emotions for instance.

It shows that some models can have a deep semantic understanding of human written text, even though there is still much work to be done in this field.

The final challenge when dealing with textual data is their temporal and sequential aspect [8]. Indeed, a text is composed of many words that are all related to each others. A word or sentence taken outside of its original context loses its meaning and information. For instance the sentence "Finally, he had a wonderful idea" shows admiration and positiveness, whereas "He finally had an idea, wonderful" could be interpreted as sarcasm or irony. The position of words can change the whole meaning of a text.

The entire document must therefore be taken as a sequential data, with each bit being related to the previous ones. This makes it harder for regular models to deal with such problems, as the relationships between words must be taken into account.

Historically, recurrent neural networks and their LSTM variant [12] have been used to deal with textual data. They are indeed built to process sequential data, and understand the context of the words pretty well. Unfortunately, RNN and even LSTM still suffer from the vanishing gradient problem when dealing

with long texts.

Since 2014, CNN have also been used to classify text [1] [2]. As shown in Fig 2, a text can be represented as a matrix, with each line being a vector representation of a word. The convolutional layers can then take in several lines at a time, which enable the network to process several words at the time. This improves the network's understanding of the context behind each word. Unlike RNNs, CNNs can get the context of a word by processing the word before it and the word after it at the same time.

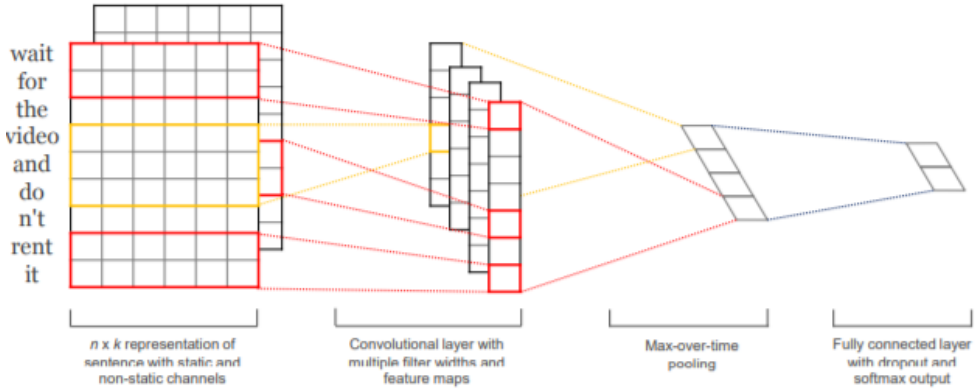

Figure 2: Text classification using CNN [1].

Until 2017, text classification was mostly performed using either RNN, LSTM or CNN [12] [1] [2]. Those models were able to handle the sequential nature of the data and to draw meaning from the relationships between words. The preprocessing of the text using padding, truncating and embedding techniques [4] [11] [5] allowed for the projection of each text into a fix-sized sequence of fix-sized vectors.

2.2 Transformers and Attention

In 2017, the paper *Attention Is All You Need* [13] introduced a new neuronal unit: the Transformer. Transformers are complex independent units first introduced for text translation. For any given input, the transformer will give an output. Depending on the desired output, a transformer can be optimized as a regular neural network using gradient descent.

The architecture of a transformer is shown in Fig 3. We can see that the model is divided into 2 units: the encoder (left) and the decoder (right). We will mainly focus on the encoder here, as it is used by BERT [9], DistilBert [14] and ALBERT [15]. More details are available in the original paper [13].

The transformer described in the article is composed of $N = 6$ encoder layers, each composed of 2 sub layers: a multi head attention block and a simple feed forward block. It also includes $N = 6$ decoder layers, but with 3 sub layers each as shown in Fig 3. There are several shortcuts in the architecture. Those allow for an easier flow of information mainly, as well as for inclusion of residual information from before the attention operations.

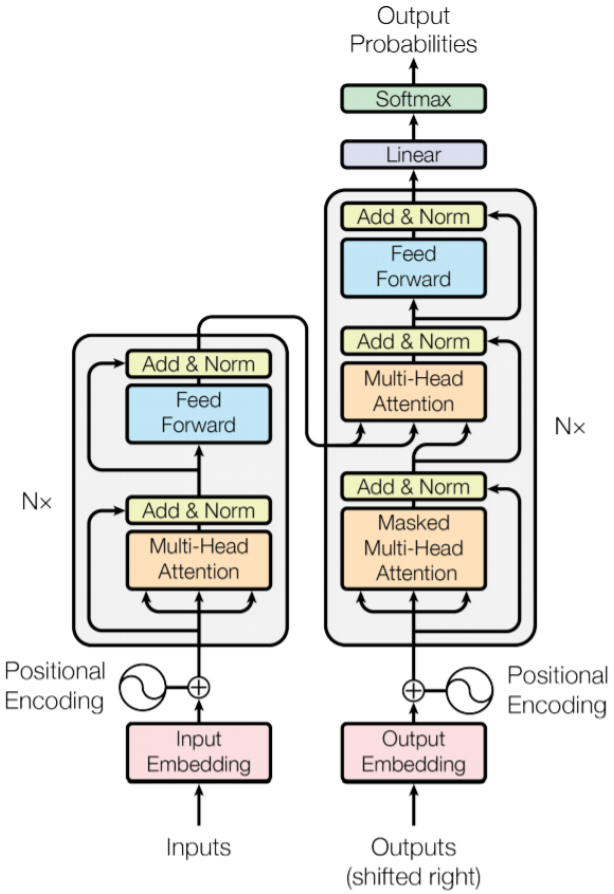

Figure 3: The Transformer - model architecture [13].

In order to take into account the position of the words in the text, words are given a positional encoding. This is a vector representing their position using a cyclic function:

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

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

with *pos* the position of the word and i the dimension. The positional encoding vectors are summed with the embedding of the words, adding contextual information to it.

The encoded vectors then enter the multi-head self-attention block. The Attention mechanism aims to find the context of each word. Let's say we have the sentence "The animal didn't cross the street because it was too tired", what does the "it" refers to? Any sane human could answer that "it" refers to "The animal", but this task would be difficult for an algorithm.

Self-attention allows to concurrently look at the relation between each word in the sequence. Unlike RNNs, Attention does not take each word individually but the entire sequence as a whole. An Attention

score is generated between each word to define how related they are. In our example, "it" would have a high attention score with "The" and "animal" and a low one with "because" whereas a RNN would likely give more importance to the "because" since it is closer to "it".

The self attention scores are computed using a key, value and query vector. For an input word x_i, the abstract vectors q_i, k_i and v_i are generated by multiplying the embedding of x_i with the matrices W_Q, W_K and W_V. Each word in the sequence has its own q_i, k_i and v_i. The self attention score for x_i regarding x_j is $z_{ij} = softmax(\frac{q_i \times k_j}{\sqrt{d_k}})v_j$, and the output of the self attention block for x_i is $z_i = \sum_j z_{ij}$.

Since we are dealing with a sequence of words, we can define the inputs as X where each line corresponds to a word. Using W_Q, W_K and W_V, the abstract matrices Q, K and V can be computed for the sequence. The self attention scores Z can then be computed as a matrix:

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

In a transformer with **multi head attention**, the self attention score is computed 8 times using different matrices W_i^Q, W_i^K and W_i^V. The outputs of all the self attention head Z_i are concatenated and the output of the entire self attention block is $Z = concat(Z_i) \times W_O$ with W_O a matrix used to weight the output. Z represent the weighted attention scores of the input sequence. Multi head attention enable a deeper semantic understanding of the attention mechanisms at play.

The output of the multi head self attention block Z is then normalized and fed into a feed forward neural network as in Fig 3, before being normalized and output of the encoder.

As shown in Fig 3, the decoder is composed of the same blocks as the encoder. The only difference resides in the second multi head attention block, which takes 2 sequences as input instead of one. Indeed, the decoder takes its own output as input to try and predict the next best output, so the second attention head has to handle the output of the encoder as well as the predicted output of the decoder.

If the input is "J'aime les tacos", the decoder has already predicted "I love" and the desired output is "I love tacos", the second attention block will compute the attention scores between the representations of "J'aime les tacos" and "I love" in order to, hopefully, output the representation of "tacos".

In order to get the best weights possible (including the best W_i^Q, W_i^K and W_i^V for the attention heads), transformers can be fully trained on huge datasets. In the article, the transformer was trained on 36M sentences for english-french translation. For a given french sentence, the transformer was trained to output the correct sequence corresponding to the english translation. Dropout, batch normalization and smoothing were used as regularization.

While the complexity of a transformer is $O(n^2 d)$ due to the attention layer, GPU can process all the computation required in parallel, which makes them significantly faster than RNN which required

sequential treatment. Computing transformers is faster even if the complexity is theoretically higher thanks to the parallel processing.

Transformers showed state of the art results for several tasks like translation, sentence prediction... On top of being efficient, they are able to grasp semantic relationships across language even if the words do not share the same position in their sentences. They also are efficient to deal with long term dependencies, as the attention mechanism enable such relationships to be discovered.

2.3 BERT model

In this subsection, we introduce the state of the art model BERT. BERT, which stands for Bidirectional Encoder Representations from Transformers, is a model created by Google in 2019 for general NLP tasks [9]. It is currently used by the search algorithm of Google and Google Translate in over 70 countries.

BERT aims to use the advantages of transformers to generate a multitasks language model. BERT is currently considered to be one of the best general text processing model [9] [14] [15].

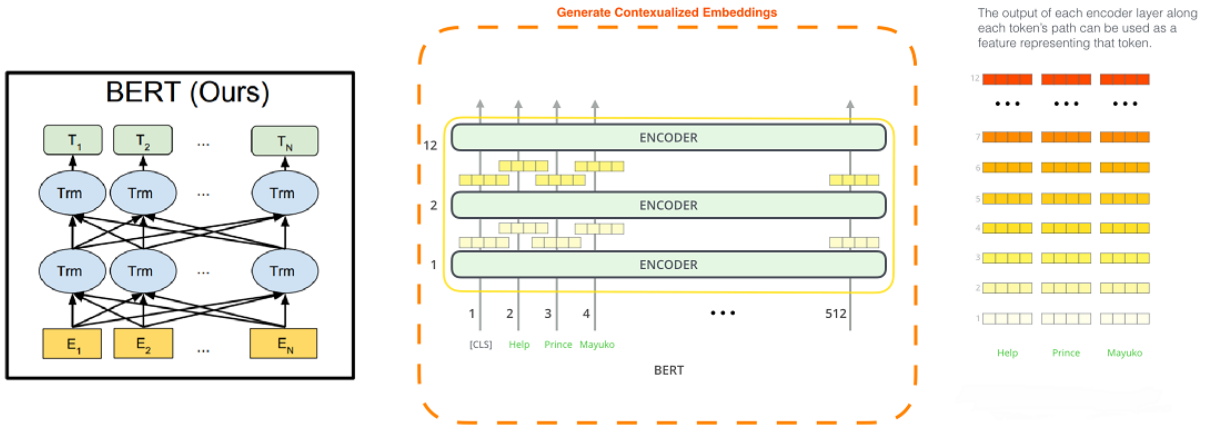

Figure 4: Simplified architecture of BERT [9] (credits to Jay Alammar).

As shown in Fig 4, BERT is entirely made of transformers. To be more specific, BERT is composed of encoders stacked above one another. For a given input sequence, each encoder will output an abstract representation as shown in the previous subsection. This abstract sequence is then fed to the next encoder, which treats it as an input sequence and output another abstract sequence representation, etc...

Each encoder layer represents an additional level of abstraction, which enables BERT to efficiently model the semantic of a language. As with transformers, each layer of BERT processes the entire input sequence at once rather than sequentially. BERT is therefore "bidirectional", although "non-directional" would be more accurate. It can learn the context of words using the attention mechanism.

The model itself is available in two different sizes. $BERT_{base}$ has 12 encoder layers (as shown in Fig 4), a hidden size of 768 (size of the output of the encoder for one word), an intermediate size of 3072 (the

dimensionality of the feed forward network in the transformers), 12 self attention heads per attention block and 110M parameters in total, while $BERT_{large}$ has 24 layers, a hidden size of 1024, 16 self attention heads and 340M parameters in total [9].

Note that BERT can only handle sequences of length up to 512. Longer sequence will have to be truncated and shorter ones will be padded. We will only focus on $BERT_{base}$ in this project, as $BERT_{large}$ is too heavy to run with my resources.

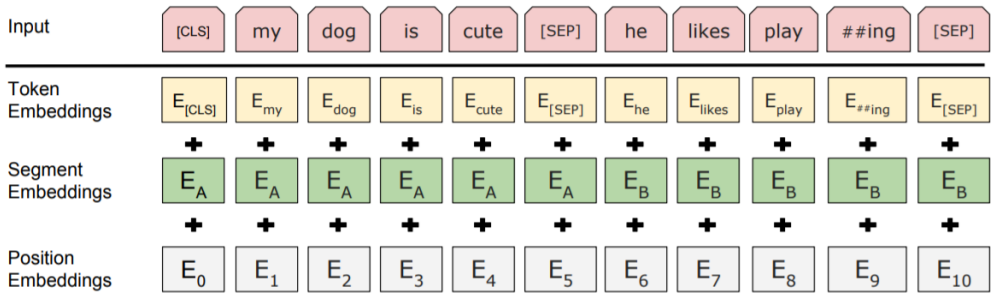

Figure 5: BERT input procedure [9].

Before being fed to the encoder layers, the input sequence is embedded using WordPiece (as shown by the "Token Embeddings" in Fig 5) and special tokens are added to mark punctuation, unknown character, etc... A special classification token ($[CLS]$) is added at the beginning. A positional embedding similar to the one used by regular transformers is also added, as well as a segmentation embedding to recognize words belonging to the same sentence.

The first output of the final encoder layer, which corresponds to the classification token ($[CLS]$), is used as an information aggregate for classification tasks. It is shown in Fig 7 and Fig 6 as $C \in I\!R^H$ with H the hidden size [9]. The final hidden vector (output of the last encoder layer) for the ith input token is $T_i \in I\!R^H$ as shown in Fig 7 and Fig 6.

A specificity of BERT is that it was designed for several NLP tasks. It can indeed be used for classification, next sentence prediction or even translation. As shown in Fig 6, BERT can be adapted depending on a specific task.

For classification problems, a softmax layer is added on the final hidden vector C which corresponds to the classification token and is used during training for classification purposes (more on that later). For Question answering tasks, additional layers are added on the final hidden vectors corresponding to the paragraph in which to search for an answer...

Despite being massive, BERT can be pre-trained to improve performance and reduce the quantity of data necessary for a specific task. This pre training takes the form of 2 distinct unsupervised tasks.

The first pre training used by BERT is the *Masked LM*. It consists in masking a word in the input

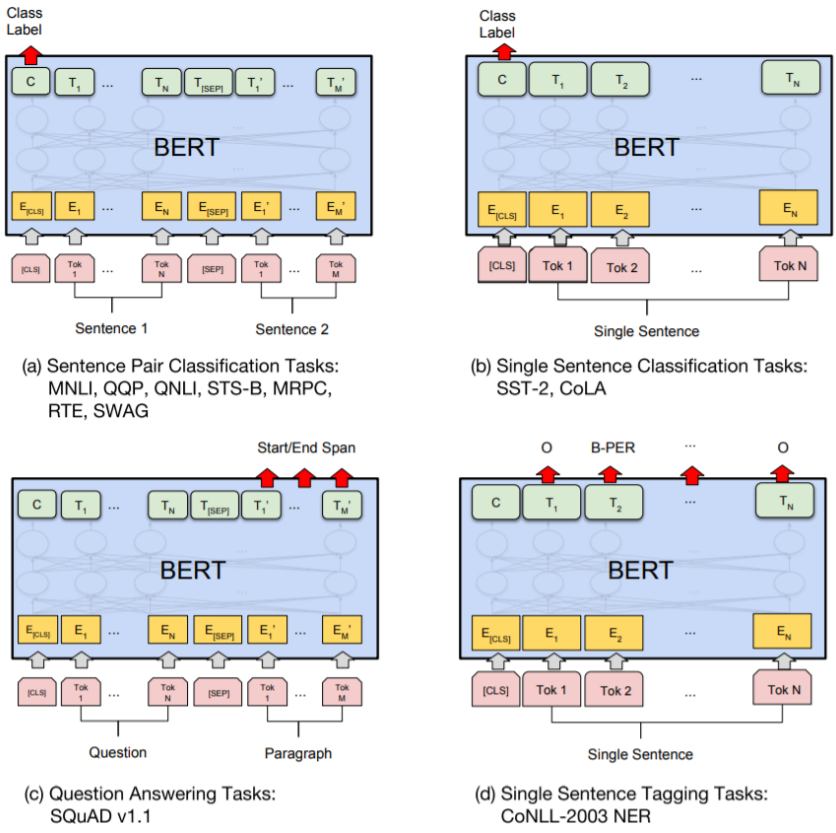

Figure 6: Fine tuning of BERT for different tasks [9].

sentence by replacing it with a special token ($[MASK]$) and training BERT to predict it using its context (the rest of the sentence). The encoder layers are optimized to output the vector representation of the masked word. This enables the model to learn the relationship between words.

An additional softmax layer must be added on top of the final hidden vector corresponding to the masked token. The loss function only takes into account the final hidden vectors for the masked tokens and ignores the ones for the non masked words as illustrated in Fig 7.

For this task, 15% of the input tokens are masked out. 80% of the 15% is replaced by a $[MASK]$ token, 10% by a random token and the last 10% is left with the correct token. The purpose of this is to bias the representation towards the actual observed word. Empirical results confirmed that this strategy was better than simply applying a [MASK] token every time.

The other pre training task is next sentence prediction. 2 sentences A and B are extracted from a corpus and fed to BERT. In 50% of the cases, B will be the sentence following A in the text. The rest of the time, A and B are just selected randomly.

The goal is for BERT to guess when B is the next sentence or not. A classification layer is added

on top of the classification hidden vector C as shown in Fig 7 and this classification layer is used during training to predict if sentence B belongs with sentence A or not.

Despite the simplicity of the task, the researchers showed that it improved the performance for several tasks like question answering and natural language inference [9].

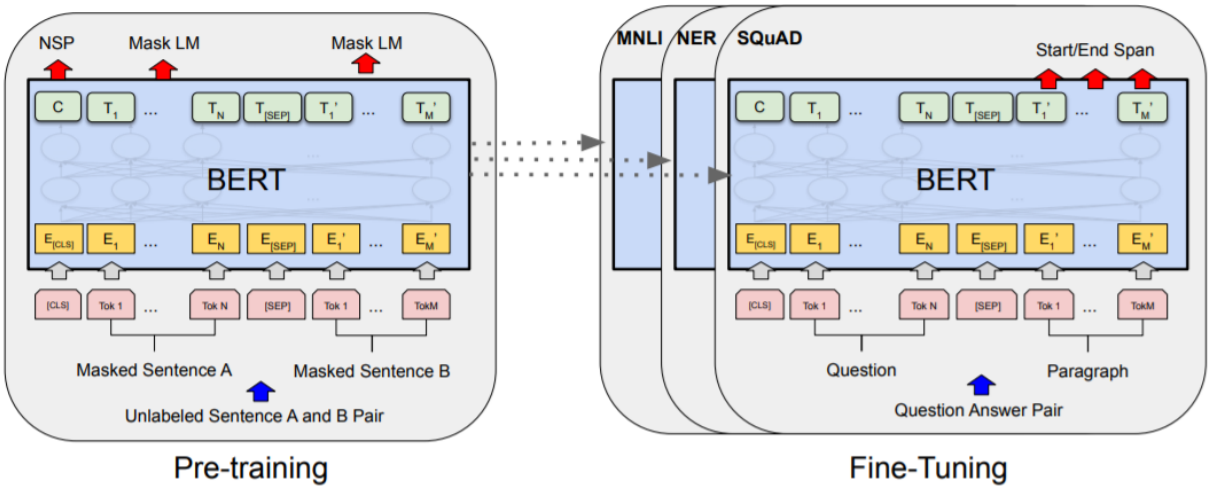

Figure 7: BERT pre-training and fine tuning [9].

BERT is pre trained using the entire wikipedia database (2500M words) as well as BookCorpus (800M words). It is worth noting that the bidirectional masked LM converges slower than left to right prediction, since only 15% of the words are predicted in each batch. However, the bidirectional training outperforms the left to right one after a few steps.

BERT still requires fine tuning in order to adapt to a specific task. Pre training BERT on huge datasets makes the network learn semantic meanings and it can then be fine tuned on a relatively small dataset and use the knowledge from the heavy pre training to fill the gaps.

A pre trained BERT can be fully fine tuned for a specific tasks by adding the necessary layers (softmax layer on C in a classification task for instance) and training the model with supervised data. In this end to end training, all parameters are fine tuned. Fine tuning is described as "inexpensive" by the researchers [9]. They show that BERT can be adapted to virtually any NLP task with minimal fine tuning.

The researchers also showed that the "benefit of depth" is clearly visible in BERT, as $BERT_{large}$ significantly outperforms $BERT_{base}$ (340M vs 110M parameters) on all NLP tasks (82.1% average accuracy VS 79.6% on 11 benchmark NLP tasks [9]). Moreover, the performance also improves as the number of steps increases, provided the model has enough data of course. $BERT_{base}$ got a 1.0% accuracy gain after 1M steps when compared to 500K steps [9].

Both $BERT_{large}$ and $BERT_{base}$ show state of the art results. They significantly outperform their competitors on the GLUE benchmark [9]. On average, the researchers reported a performance of 82.1 for

$BERT_{large}$ and 79.6 for $BERT_{base}$, while the best competitor only had 75.1.

2.4 DISTILBERT model

According to its team [9], BERT obtains amazing results on several benchmark and is THE state of the art model for text processing. Although different studies [9] [15] [14] and a certain notoriety tend to confirm this, BERT still has a major downside: its weight.

BERT is made of 110M parameters for the base version. While running it on a server with the appropriate resources may not be a problem, many students, amateurs or even small start-ups do not have access to powerful servers and therefore cannot use BERT.

BERT may also be poorly suited for specific tasks with low resource constraints. BERT cannot be used on mobile devices as it would consume too much battery and processing power for instance. Moreover, the energy consumption of such model does not align with the goal of reducing the environmental footprint of our systems.

DistilBert was created in 2019 with the aim to create a lighter and faster BERT [14]. It is therefore based on BERT and uses a similar architecture, but with a reduced number of layers.

DistilBert only has 6 encoder layers instead of 12 and a total of 66M parameters. The hidden size, remain the same however, as the team reports it has a minimal influence on the performance of the model. Modern linear algebra frameworks are also used to further optimize the operations done by the transformers. The architecture of DistilBert is summarized in Fig 8.

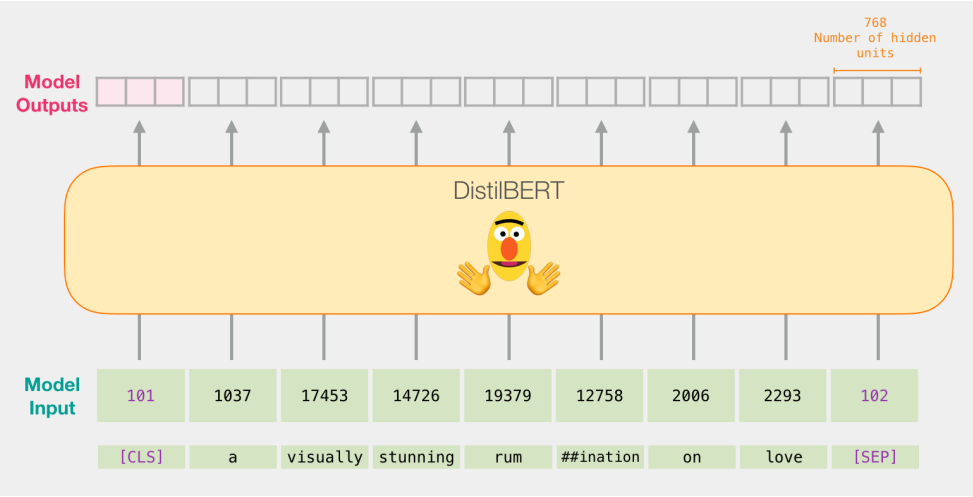

Figure 8: DistilBert model [14] (credits to Jay Alammar).

Before training, the parameters of DistilBert are initialized using those of BERT. Since both models do not have the same size, DistilBert is initialized by taking one layer out of two from the pre-trained

BERT.

DistilBert then follows the same training procedure as BERT. It is pre-trained on the same datasets (more than 3 billion words) using knowledge distillation. As shown in Fig 9, knowledge distillation is a compression technique in which a compact model - the student DistilBert - is trained to reproduce the behavior of a larger model - the teacher Bert. DistilBert is pre trained on the masked LM task only, not the next sentence prediction as it does not improve the performance of DistilBert according to its team.

Instead of solely being trained to minimize the cross entropy on the masked LM task, DistilBert is also trained to obtain the same output probabilities as BERT. This allows DistilBert to gain the generalization capacity of BERT.

More specifically, DistilBert is trained to minimize the sum of 3 loss functions $L_{mlm} + L_{cos} + L_{ce}$. DistilBert uses a cross entropy loss L_{mlm} on the masked LM pre training task. It also uses a cosine embedding loss L_{cos} which apparently align the "directions of the student and teacher hidden states vectors" [14]. The final loss is a distillation loss over the soft target probabilities of the teacher: $L_{ce} = \sum_i t_i * log(s_i)$ with t_i a probability estimated by the teacher and s_i by the student.

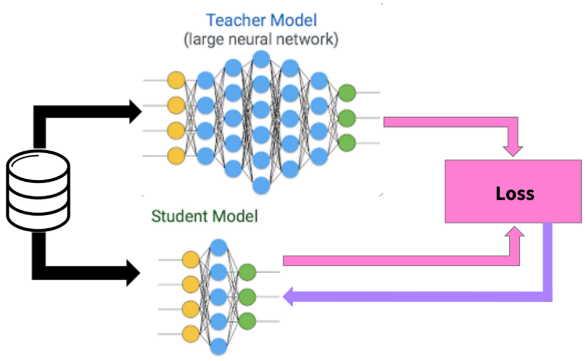

Figure 9: Knowledge distillation procedure.

Once pre-training is over, DistilBert can be fine tuned for a specific task exactly like BERT. DistilBert obtains an average score of 77% on the GLUE benchmark, with $BERT_{base}$ having 79.5%. As the article puts it, DistilBert shows that "...it is possible to reduce the size of a BERT model by 40%, while retaining 97% of its language understanding capabilities and being 60% faster".

While the use of 3 training loss may appear unnecessary, the researchers showed that the 3 losses and the initialization using BERT weights were all increasing the performance of DistilBert. Using an ablation study, they showed that the average score of DistilBert dropped by 2.96% if L_{ce} was removed. Removing L_{cos} reduced the score by 1.46% and removing L_{mlm} only by 0.31%. It shows that the losses do not have the same importance for the performance of the model. L_{ce} is more important than L_{cos} which is more important than L_{mlm}. When using all 3 losses but random weights at initialization, the performance dropped of 3.69%, which highlight the importance of using BERT's weights for initialization.

2.5 ALBERT model

ALBERT is another model based on BERT and published in 2020. As for DistilBert, the motivation behind ALBERT is that BERT needs too much training time and has a high memory consumption. ALBERT aims to create a lighter and faster BERT-like architecture with a lower memory consumption and a reduced training time [15].

ALBERT follows the same architecture as BERT. Unlike DistilBert, which has a reduced number of layers, ALBERT still possess 12 encoder layers, 12 attention heads per attention block, an intermediate size of 3072 and a hidden size of 768. It therefore has the exact same architecture as $BERT_{base}$.

In order to increase speed and reduce memory consumption, ALBERT relies instead on 3 main technical choices: the factorization of the embedding matrix, the use of repeating layers (weight sharing), and finally the use of an inter-sentence coherence loss.

In Bert, the WordPiece embedding size E is tied with the hidden layer size H: $E \equiv H$. The team argue that this is sub-optimal for 2 reasons. Firstly, embeddings are meant to learn context independent representations while the hidden vectors are meant to learn context dependent ones.

In BERT, the one hot vectors of size V (with V the size of the vocabulary, i.e $V \simeq 30000$) representing the embedded input are directly projected into hidden vectors of size H. This ties the embedding with the context dependent projection into hidden vectors. It would make more sense to clearly separate those two steps from a modeling perspective. Indeed, the embedding is supposed to carry less information as it is context independent, and we should have $H \gg E$.

From a computational point of view, projecting the one hot vectors of size V directly into hidden representations of size H also implies the need for a matrix of size $V \times H$, which, considering the size of V and H, can results in billion of unnecessary parameters.

The researchers decided to separate the embedding from the projection into hidden vectors. They decomposed the embedding parameters into 2 smaller matrices [15]. Instead of projecting the one-hot vectors directly into the hidden space of size H, they are first projected into a lower dimensional embedding space of size E, and then projected into the hidden space. This reduces the complexity from $O(V \times H)$ to $O(V \times E + E \times H)$, which is significant if $H \gg E$.

The researchers used an embedding size of 128 for ALBERT, i.e $E = 128$ (and $V \simeq 30000$ and $H = 768$).

The second technical choice of ALBERT is to use repeating layers. In a neural network, it is possible for some weights to be shared and tied. This means that they will have the same values, and the modification of one will also change the other. This techniques has been shown to reduce the complexity of deep architectures while having a minimal effect on global performance [15].

Repeating layers are a particular type of weight sharing. It implies sharing all the weights across layers. Instead of simply sharing the weights of the feed forward network of each transformer for instance, each encoder layer in BERT will have the same weights as the previous and next one. It can be simply modeled as going through the same encoder layer several times, with the output being used as input for the next iteration.

According to the team, repeating layers are effective to smooth the transition between layers, meaning that the L2 distance of the output of each layer are more similar than the ones of BERT.

Using repeating layer also has a huge advantage regarding training: the number of actual parameters is reduced to that of one layer only. Thanks to this technique, ALBERT only has 12M parameters. While this definitely helps to lower the training time, it does not actually increase the speed of the network. Indeed, a sample still has to go through the same numbers of layer, the difference being that it will go through the same layer 12 times instead of 12 different layers.

The last technical choice for ALBERT is to use an inter sentence coherence loss during training. This coherence loss is designed to be used during pre training on a modified version of the next sentence prediction task.

The original task of next sentence prediction was qualified of unnecessary by the team of DistilBert, and the team of ALBERT calls it easy. To train ALBERT, they design a different multi sentence task. Instead of predicting if a sentence B is following a sentence A, the new task is about predicting if two sentences are in the right order. This sentence order prediction loss uses as positive examples the same technique as BERT (two consecutive segments from the same document), and as negative examples the same two consecutive segments but with their order swapped.

The team showed that training ALBERT on this SOP loss still enabled it to perform well on the next sentence prediction task, while the opposite wasn't true. Using this new task consistently improved downstream task performance for multi-sentence tasks.

For training, ALBERT is pre trained using masked LM and sentence order prediction on the same corpus as BERT. It does NOT use distillation like DistilBert, and is directly pre-trained on the corpus. It can then be fine-tuned for a specific task by adding the necessary layers on top of the repeating layer.

Note that the researchers experimented ALBERT with different architectures, weight sharing methods and embedding factorization. Some architectures were very large and had great performance. For instance, the ALBERT-xxlarge had an embedding size of 128, a hidden size of 4096, 12 hidden layers, 64 attention heads and an intermediate size of 16384 for a total of 235M parameters. According to the team, this huge model performed better than $BERT_{large}$ [15], getting a performance gain of 3.5% on some GLUE tasks while being 3 times faster.

Since we aim to study rather light models in this project, this variant of ALBERT will not be taken into account and we will only study the one with 12M parameters detailed in this subsection.

The studied model already performed surprisingly well. Indeed, the use of $E = 128$ with fully shared weight allows ALBERT to score 80.1% on some GLUE tasks while $BERT_{base}$ (reminder: 110M parameters) gets an average score of 82.3% on the same tasks. ALBERT is also 20% faster than $BERT_{base}$.

While the ALBERT we study is not the more performing, it has a low complexity while maintaining as much accuracy as possible. More variants and their reported performance are available in the original paper [15].

Overall, the factorization of the embedding reduces the number of parameter of the model and thus increases its speed. On the other hand, the use of repeating layers reduces the training time but not the forward pass time. The sentence order prediction task improves the pre training compared to BERT. ALBERT is therefore faster to train, faster to predict (thanks to the embedding factorization) and better trained. While having a very reduced number of parameter and being faster than BERT, it apparently still performs amazingly well.

3 Methodology

As shown in the previous section, BERT, DistilBert and ALBERT all seems to have their own strengths and weaknesses. However, some claims in the papers are bringing new questions. DistilBert claims to be one of the best performing model [14] but is not mentioned in the more recent paper introducing ALBERT [15]. Moreover, the models are tested on a specific benchmark that favors general language model rather than classification. A fair and reproducible comparison of the 3 models is needed to determine their respective potential in text classification

In this section, we will introduce the different components of the methodology used during the project. This includes the databases and metrics used to evaluate the different models, as well as the experimental protocol and the limits of this project.

It aims to provide exhaustive information regarding the course of the project to ensure its reproducibility and impartiality.

3.1 Databases

To have a meaningful comparison of the different models, the first step is to choose the databases to use for our benchmark.

Since text processing is a popular domain of research, there are many databases available online for text processing. Many of those datasets are quite old and made of short texts.

Considering the number of different tasks in text processing, some of those datasets are made for specific tasks other than classification and are irrelevant for this project.

However, text classification probably is the text processing task for which the number of dedicated datasets is the highest. Indeed, classification is the oldest text processing task and one of the simplest to understand, as it is similar to other types of classification. Moreover, there has been a lot of interest for this task over the years.

Many of the datasets available for text classification are composed of short sequences of text but have a high number of samples. While training heavy models with short sentences may affect performance, the quantity of samples should make up for the size of individual samples.

Short sequences are also interesting to study for several reason. Firstly, it will speed up the training and reduce the resources consumption for evaluation and training. Considering the limited resources at my disposal, this is a quite attractive argument.

Indeed, the shorter the sequence, the easier it will be to process the attention scores in the transformers. Indeed, a transformer outputs the same number of hidden vectors as the number it had as input. Moreover, the attention mechanism has a complexity of $O(n^2)$ with n the length of the sequence. Reducing the length of the input sequence is therefore theoretically beneficial regarding performance. Note that

some software optimization can also be added, thus lowering even more the computation time.

In addition, short sequences are more representative of the majority of the textual data used today. Although some texts can be huge and contain thousands of words, most of today's textual data is made of messages, social media commentaries, mails... etc

Working on short sequences is therefore more representative of the type of data available in real life applications. Such applications are usually bound to some specific areas, like sentiment analysis or classification of short memos. Such work can be automated, but giving a deep analysis of a long and complicated text will probably remain a human-only work (for some times at least). Focusing on short text containing, supposedly, no deep philosophical arguments is more representative of real life applications.

Two datasets are used in this project. They are representative of real life applications and are both made of short and quite straightforward sequences for the aforementioned reasons:

- Large Movie Review Dataset "IMDB" [6]: This is a dataset for binary sentiment classification. The core dataset contains 50,000 annotated online movie reviews. The overall distribution of labels is balanced (25k pos and 25k neg). In the entire collection, no more than 30 reviews are allowed for any given movie to prevent any bias in the data. A negative review has a score ≤ 4 out of 10 on imdb.com, and a positive review has a score ≥ 7 out of 10. Thus reviews with more neutral ratings are not included.

- AG News DataSet [7]: This is a dataset for topic classification using short english news articles annotated into 4 different mutually exclusive categories. It is composed of news description and their classes from AGWeb.com. The news are classified into different categories that are World, Business, Sports and SciTech. It is a human-labeled dataset with 127600 balanced items, 31900 for each class. Each item is a small straightforward news description.

It is worth noting that the samples are significantly shorter in the AG News dataset (average length=44 words) compared to IMDB (average length=292 words). Nonetheless, they both contain samples commonly encountered when performing text classification.

3.2 Metrics

Choosing the right metrics is important for the project. It ensures reproducibility and the relevance of the experiments. The chosen metrics must represent the performance of each model as best as possible.

Representing the strengths and weaknesses of a model using only one metric is unrealistic, so several will be used. The metrics must also be general enough to allow comparison on both our datasets.

Text classification is a quite simple task. As any other classification task, the main metric we will use is the accuracy of each model during evaluation.

The accuracy of a model can be defined as:

$$Acc = \frac{N_{true}}{N_{total}}$$

with N_{true} the number of samples for which the model predicted the correct label during evaluation and N_{total} the number of samples in this dataset.

Although accuracy is the simplest metric used in classification tasks, it is important to note that it also requires to take into account the number of labels. Indeed, the more different labels there are, the lower the impact of chance in our data. With the IMDB dataset, the probability of guessing the correct label by pure chance is 50%, whereas it drops to 25% for the AG News dataset. Although the training of our model should make such impact barely noticeable, it can still have a minimal impact on our results.

Due to those reasons as well as to keep as much granularity as possible in our results, the accuracy will be calculated on each dataset separately.

The other common metric used during any deep learning task is speed. Speed can be separated into 2 different metrics : the training speed and the evaluation speed.

The training speed is the speed during the training of the network. It can be quantified by measuring the time required to train a model for one epoch.

The evaluation speed represents the speed during the evaluation of the network (after training). It can be quantified by measuring the time required to predict the label of a given test dataset.

Marking the difference between those 2 speeds is important, as training takes more time than evaluation, but evaluation is a more interesting metric for real life applications. Indeed, The training only have to be performed once, while the evaluation/prediction is what a model is made for. Thus, it is commonly accepted that a model can be slow to train, but must be quick to predict labels in production.

While it is not a metric but rather a hyperparameter, studying the impact of the length of the samples can also be interesting. Indeed, as mentioned before, the length of the samples can affect the accuracy and both the training and evaluation speed of our models. We can therefore quantify the impact of this length on the aforementioned metrics by measuring the metrics for a given sample length.

Even more granularity can be added by training a model using a certain length, and then testing the generalization capabilities of the model on samples of different length.

More explanations will be given in the results section.

Using mainly accuracy and speed, we can provide an impartial comparison of our models with some granularity regarding the impact of the length of the samples.

3.3 Environment and Protocol

The experimental protocol must be made explicit to ensure impartiality and reproducibility. We will explain how we intend to setup our models and measure their performance.

Considering the size of our models (up to 110M parameters), we must use an external environment as a single computer may not handle such model very well or take too much time doing it. For this project, the environment used will be Google Colab with a GPU runtime. According to Google, this enables us to access up to 12GB of RAM and several Tesla K80 NVIDIA GPU. All our experiments are performed in this environment and the code we used is available on the Colab plateform.

The BERT, DistilBert and ALBERT models are too complicated to be implemented and trained from scratch. In this project, we use the *transformers* package from HuggingFace which contains several pre trained state of the art models, including the ones we use in this project. The three models used during this project come from this package, as well as their respective pre trained weights. I used their respective TensorFlow (TF 2.1.0) implementations as I am more familiar with it, but PyTorch implementations are also available.

Regarding the experimental protocol, we use the pre trained weights available in the *transformers* package for initialization and we then fine tune the models using our datasets. In order to do so, our datasets are separated into training, validation and testing datasets, with a random 20% of the original datasets being used for validation, another 20% for testing and 60% for the fine tuning.

The validation dataset is used to select the best epoch during fine tuning, and the test dataset to provide the final accuracy score of the fine tuned model.

We were unable to perform an extensive search for the best hyperparameters due to time and resources constraints. For each combination of model-dataset, we fine tuned the model for up to 3 epochs maximum, using a learning rate of 3.0E-5 and the ADAM optimizer algorithm.

We selected the epoch with the best validation accuracy for each model and used the test dataset to get the final accuracy score. For the training speed, we take an average of each epoch training time, and the evaluation speed is the time taken for the prediction of the test dataset's labels.

The samples are embedded using the WordPiece tokenizer provided in the *transformers* package. We perform the aforementioned protocol for several different max length of the samples: 32, 64, 128 and 256. Due to resource constraints, length 512 was not used as it made $BERT_{base}$ too heavy even with Google Colab.

Once the combinations model-datasets have been tested using this protocol for the different samples

lengths, we report the results, which are available in the next section.

3.4 Limits of the methodology

The methodology used for this project aims to provide exhaustive and impartial information for all models involved. Unfortunately, it is not perfect and has some disadvantages that must be highlighted.

The first obvious criticism that can be made is the reliance on exterior implementations for the models BERT, DistilBert and ALBERT. Using the transformers package instead of the "original" implementations may be an error as we cannot verify the quality of the implementations compared to the ones of their original papers. Our results are completely dependent on the quality of the implementations.

However, the Hugging Face company which maintain the *transformers* package is reliable. They are specialized in complex neural architectures and they are the company responsible for the original implementation of DistilBert [14]. The implementation of DistilBert used for the project is made by the original authors. Moreover, even Google acknowledges their work and provides models through their open source package, meaning that the BERT and ALBERT implementations are also very reliable.

Although the models are reliable, we were unable to provide an extensive hyperparameter optimization during this project. Thus, it is likely that our models have sub-optimal performance. While this is mainly true, our results show that increasing the number of epochs during fine tuning is unnecessary. In addition, it is impossible to find THE best hyperparameters and doing a more extensive search for better hyperparameters would cost time and resources that we do not have.

On top of that, the goal of this project is not to find the best hyperparameter combination, but to study the "trend" of each model. While this is not exhaustive, it provides the basic necessary information to compare the different models and select the best suited one for a specific task.

We can also argue that our datasets are limited to certain specific domains (sentiment analysis and news classification) and that our results may not be representative of a general trend of each model. It is correct that we cannot confirm our results are valid for other types of text classification tasks. However, the general understanding of textual data that our models use can easily be applied to a wide variety of tasks as shown in their respective original papers [9] [14] [15]. Hence, if their general understanding enables them to perform many NLP tasks, it surely also enables them to perform on many different classification tasks with coherent results.

While it is true that our datasets are quite straightforward and "simple", it still enable us to show the "trend" of each model. It also matches real life application data.

Finally, we have the problem of relying on an exterior environment (Google Colab). This can prove to be a problem, as we do not have access to the resources management of Colab. It is possible that our training and evaluation speed measurements are altered.

Indeed, Colab is a free to use platform where many people can connect while the available resources remain the same. To optimize Colab, the resources are shared among the users depending on their needs. It means that the RAM and GPU resources available for our project can be reduced when the platform is being overloaded.

While it does not affect the accuracy of our model, the training and evaluation speed can be affected by this. Unfortunately, we have no means to correct this to my knowledge. Since I do not have the resources to perform the experimentation myself, I must rely on Colab despite the fact that the speed measures may be wrong.

Thankfully, the results we got matched the ones we expected. Moreover, the training speed results were taken as an average over several epochs (lasting hours), so we would normally see any abnormal variation in the training time.

While our method is not perfect, it provides the necessary results for this project and enables us to get trend information for each model. We must keep the limits of our protocol in mind when discussing the results of the experiments.

4 Results of the benchmark

Using the aforementioned databases and protocol, we present the results of the project in this section. As mentioned in the methodology, we obtained results for each possible combination of our model with our datasets. We also add granularity to those results by providing a limited study on the impact of the length of the samples regarding the performance of the models.

A top down approach seemed to be the best way to present our results. In this section, we will first present the global accuracy and speed for each model. We will then add granularity by studying the impact of the length of the samples on the accuracy and speed of each model.

4.1 Global results

When experimenting with the models, the first natural goal we had was to measure the accuracy of each model to see how they compare on each dataset.

We therefore performed a small hyperparameter optimization mainly focused on finding the right number of training epoch (up to 3 maximum as mentioned in the Methodology section) and the best length for the samples (up to 256 words).

The hyperparameter search showed that searching for more than 3 epochs was useless. Indeed, the lowest validation loss was usually obtained at the first epoch for almost all the combination of models and datasets.

The search for the best length of samples showed that, in almost all cases, an increased length meant a higher accuracy. The best accuracy was almost always obtained by taking samples of size 256. However, the length of the samples proved to be interesting to further study in depth, as we will see in the rest of this section.

Dataset	IMDB dataset		AG News dataset		Both datasets
Metrics	Accuracy	Length of samples	Accuracy	Length of samples	Avg. accuracy
Model					
$BERT_{base}$	0.91	256	0.934	64	0.922
DistilBert	0.90	256	0.931	256	0.916
ALBERT	0.88	256	0.922	256	0.901

Table 1: Global accuracy of the models.

The accuracy of the model is reported in Table 1. As we can see, BERT obtains a higher accuracy than DistilBert and ALBERT on both datasets. DistilBert also obtains a better accuracy score than ALBERT on both datasets. On average, a clear trend shows that BERT is more accurate than DistilBert, which is

itself better than ALBERT.

This was expected, as DistilBert and ALBERT were designed to have a lower accuracy than BERT at the benefit of an increased speed. Seeing BERT perform better is therefore not a surprise. What is interesting is the fact that DistilBert beats ALBERT, even though ALBERT is more recent than DistilBert. Since ALBERT has less parameters than DistilBert, we can argue that the performance is closely tied with the complexity of the model, as models with more parameters show better results on both datasets.

The results in Table 1 also show that the models perform better on the AG News dataset than on IMDB. This is somehow unexpected, as IMDB only has 2 labels whereas AG News has 4. However, the average length of a sample is 44 words for the AG News dataset, and almost 200 for IMDB. Shorter sequences may be easier to process than long ones, but the difference in results could also be linked to the nature of the data. Indeed, AG News samples are straightforward news articles, whereas the ones of IMDB are human commentaries. Sentiment analysis is probably more complex than processing simple news articles.

We can also see that, while most of the best accuracy scores are obtained with samples of length 256, the best accuracy for BERT with the AG News dataset is obtained using samples truncated or padded to 64 words. This is not surprising since the average length of the samples is of 44 words, but it is strange that the other models still performed better with longer sequences. The longer sequence probably helps them handle the few longer samples there are, and perhaps the complexity of BERT makes it able to handle shorter truncated sequences regardless.

Despite this, all models obtain a quite similar accuracy. Indeed, there is less than 2% difference between the accuracy scores. While complexity seems to improve the accuracy, its effect appear to be limited compared to the difference of parameter numbers (110M vs 66M vs 12M). In fact, the accuracy scores were so similar for that we had to use 3 digits to present the scores on the AG News dataset in Table 1, otherwise there would be no way to distinguish the scores of BERT and DistilBert.

Regarding the accuracy only, the accuracy seems to improve with complexity but with limited effect. Although their is a clear hierarchy among the models according to our results, the difference between the models are minimal when compared to their respective complexity. DistilBert and ALBERT both perform very well when compared to the much more massive BERT.

While accuracy is the most commonly used metric to study different models, speed is also relevant. In our case, the speed of the models used in Table 1 brings very interesting information displayed in Table 2.

Dataset / Metrics / Model	IMDB dataset			AG News dataset		
	Train. speed	Eval. speed	Length/samples	Train. speed	Eval. speed	Length/samples
$BERT_{base}$	2520	220	256	1740	176	64
DistilBert	1285	110	256	1400	150	256
ALBERT	1265	130	256	2693	305	256

Table 2: Speed of the selected models (in seconds).

Note that the values shown in Table 2 are measured using an entire epoch for the training speed and the test dataset for the evaluation time. Using an average speed is therefore irrelevant as the epochs and test datasets contain different numbers of samples.

Table 2 shows the downside of having huge models. BERT is indeed the slowest model for the IMDB dataset. It is nearly twice slower than DistilBert and ALBERT for both training and evaluation.

While BERT had a better accuracy than DistilBert and ALBERT, it also is significantly heavier and thus slower than both our light models. The results in Table 2 are not surprising. In fact, DistilBert possesses only 6 encoders instead of 12, which makes it completely logical for it to be twice faster than BERT.

However, ALBERT appears almost as fast as DistilBert, which is understanding for the training time but not for the evaluation. ALBERT uses repeating layers, so the evaluation time should be closer to the one of BERT rather than DistilBert. As explained in the methodology section, this abnormality can be explained by the sudden allocation of more resources by Google Colab. It can also be caused by a certain optimization in the implementation, or is simply due to the factorization of the embedding matrix. We cannot know for sure, but the latest option is definitely a reason of the abnormality.

The results for the AG News dataset are more unexpected. The BERT model is indeed faster to train and evaluate than ALBERT for this dataset, and DistilBert is the fastest for both training and evaluation. When we first got the results, it was hard to understand how a model design to be faster than BERT could be slower...

However, there is a variable that has not be taken into consideration yet: the length of the samples. Indeed, the results are those of a BERT model chosen because of its accuracy (as in Table 2) and which was trained and evaluated using samples of length 64 instead of 256 like the other models in Table 1 and 2. The use of samples of different length is the reason for such odd results (more on that in the next subsection).

Moreover, we get the expected evaluation time for ALBERT. Indeed, it takes twice as much time as DistilBert for evaluating, which is normal considering that DistilBert has 6 encoders and ALBERT uses 12 repeating layers. ALBERT is also slower than DistilBert for training, and BERT should, with the same length of samples, be even slower than that.

Overall, those global results show us that BERT is heavier and slower than DistilBert and ALBERT for both training and even evaluation for DistilBert. BERT also has a better accuracy thanks to its complexity, but the two lighter models almost get identical results. DistilBert seems to be faster than ALBERT for both training and evaluation, while also having a better accuracy. The complexity of BERT appears unnecessary compared to the lighters models since their accuracy is similar to BERT's while being much faster.

Surprisingly, the most interesting information drew from those first results is that the length of the samples may have a huge impact on the speed of the models. BERT is indeed faster than ALBERT with samples of length 64, while having almost 10 times more parameters. It was expected for the length of the samples to impact the processing speed, but not at this scale. Thus, studying the impact of this hyperparameter may prove very interesting to select faster models and characterize BERT and its variants.

4.2 Impact of the length of the samples

In the previous subsection, the first results showed that using further truncated samples could significantly increase the training and evaluation speed of a model while not necessarily reducing its accuracy.

However, more data are needed to characterize the impact of this hyperparameter. We must study its impact on both speed and accuracy on each model with each dataset. Since we cannot study the results for every length of samples, we focus on the lengths of 32, 64, 128 and 256 as mentioned in the Methodology section. Each model is trained and evaluated with samples of the aforementioned length.

The objectives are to study the relation between the length of the samples and the speed of a model as well as its accuracy. We suspect it is possible to reduce the length of the samples to a certain extent without having too much impact on the accuracy.

Ideally, we could define an optimized length for which we would conserve as much accuracy as possible while significantly speeding up our models. To my knowledge, the impact of the length of the samples has not been well characterized yet in the scientific community.

We train and evaluate the models with the different lengths. The accuracy of each configuration is shown in Fig 10.

The first thing we can notice is that the evolution of the accuracy is not linearly dependent on the length of the samples. Indeed, it appears to follow some sort of logarithmic distribution. This implies

that the length of the samples can be reduced without loosing to much accuracy until some point.

There is a notable difference between the evolution for the IMDB and for the AG News dataset. Indeed, they both have a similar shape but the curve for the AG News dataset reaches a plateau before the one of IMDB. In practice, it means reducing the length of the samples will have a greater effect on the accuracy for the IMDB dataset than for AG News.

This difference of plateau can be explained by the nature of the data. The samples of IMDB are on average longer than those of AG News, so truncating them means loosing more information than when truncating the short sequences of AG News. To support this claim, we can see that the plateau of each curve is approximately reached when the samples are truncated/padded to the average length for each dataset. Hence why the plateau for AG News begins at 64 while the one for IMDB starts near 256.

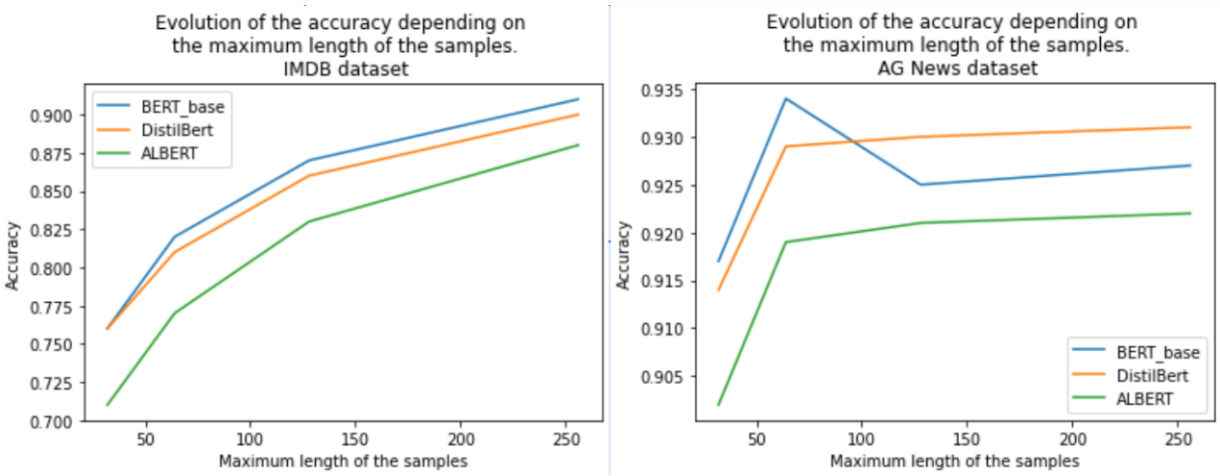

Figure 10: Relation between accuracy and length of the samples.

As for the global results, we can notice that BERT appears to be more accurate than its variants for the IMDB dataset. For the AG News dataset, BERT has the highest accuracy but only for one specific length of sample (64), otherwise it actually is less accurate than DistilBert.

BERT seems to perform better when the samples are truncated or padded to their average length. When the samples are not well padded/truncated, its performance decreases quite quickly.

For instance, BERT gets the best results for the IMDB dataset when the length used is 64 or higher, but not when the length is too short. This is not simply due to the fact that there is a loss of information when truncating the samples, as its performance also decreases with padded sequences longer than 64 for the AG News dataset. The further away from the average length of each dataset, the worse BERT performs.

On the other hand, DistilBert and ALBERT do not seem to be affected as much by this. When using padded samples longer than 64, their accuracy does not drop like BERT's but remains quite steady. However, they also loose accuracy when truncating the samples too much. In such situation, DistilBert

appears to perform as well as BERT (see the IMDB with length = 32).

ALBERT has a lower accuracy than BERT and DistilBert on both datasets and for all the different length of samples. While ALBERT was expected to be less accurate than BERT, the comparison with DistilBert is disappointing. DistilBert is indeed older (not by much but still older) and ALBERT uses more complex parameter reduction techniques than the simple distillation of DistilBert [14] [15]. It is worth noting that ALBERT has a great accuracy nonetheless, just not as good as the ones of its competitors.

DistilBert is the biggest surprise of this study. It performs really well on both datasets, beating ALBERT all the time and even surpassing BERT for short padded sequences. This is the case in the AG News dataset, and shows that the simplicity of DistilBert may actually make it more resilient and robust than BERT for unusual data.

When using truncated data, or data that are not padded too much, DistilBert is able to compete with BERT and only has a minor difference, conserving more than the 97% accuracy promised by the original article [14].

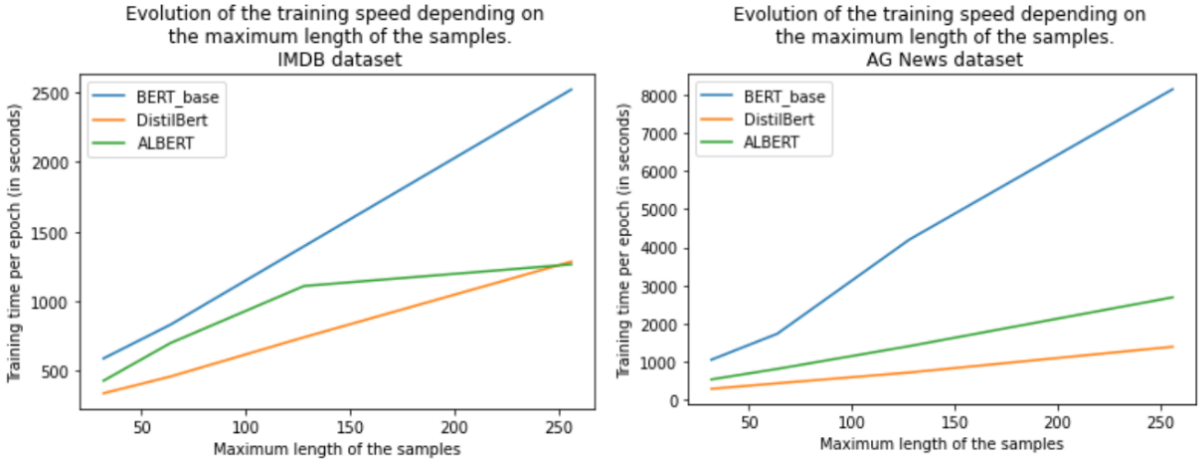

Figure 11: Relation between training time and length of the samples.

Fig 11 shows the evolution of the training speed depending on the length to which the samples are truncated/padded.

Unlike the accuracy, the training speed seems to have a linear relation with the length of the samples. This relation is stronger than between the length of the samples and the accuracy. Indeed, reducing the length of the samples can greatly increase the speed of a model.

This confirm our first intuition: the speed of a model increases significantly faster than its accuracy drops. When using transformers, we can reduce the length of the samples without significantly affecting the accuracy. It opens the door to the use of faster text classification models with little accuracy loss.

We can see why BERT was faster to train than DistilBert in our previous results. The scale at which the training time increases with the length of the samples is impressive. With a length of 64, the massive BERT was faster than the light ALBERT with a length of 256. Although BERT is slower on average and heavier than ALBERT, it can become easier to train with shorter sequences.

It shows the potential of leveraging the length of the samples to improve the speed of a model. A huge model can benefit of a huge speed-up with minimal accuracy loss simply by leveraging this hyperparameter.

The training speeds reported in Fig 11 confirm that BERT is slower than the other models for the same sequence's length. It validates our expectations and explains why BERT using a specific length was faster than other models as mentioned in the previous subsection. Indeed, BERT should logically be slower than its variants designed to be faster.

Once again, DistilBert appears to perform amazingly well. It is the fastest model for all the tested length. As expected, DistilBert is 2× faster to train than BERT for the IMDB dataset, which is due to its architecture made of 6 encoders instead of 12. For the AG News dataset, the gap is even wider. The reason of the difference between the two datasets is not a sudden acceleration of DistilBert, but a slow down of BERT.

Indeed, the training speed of DistilBert measured during this project is perfectly linear when compared against the length of the samples. On the other hand, BERT is linear for the IMDB dataset but not AG News. The training of BERT is significantly slower than expected for the length 256 and 128 when compared to 64 and 32.

This slow down can be explained by the environment. The AG News dataset features more samples and training BERT with it consumes a lot of memory and processing power. Perhaps the resources consumption was too high and Colab slowed the environment down to free resources for other users. It is also possible that we reached the limits available for a session, or that there was not enough GPUs available to compute all the training in parallel and some of it had to be done sequentially...etc

ALBERT is, once again, disappointing regarding the training speed. Although it is significantly faster than BERT, which was expected, ALBERT is slower to train than DistilBert for both datasets.

Despite being the lightest model (12M parameters) and the one with the worst accuracy, ALBERT is slower than DistilBert almost by a factor 2 on the AG News dataset! It shows that the environment may not be the only thing to blame for the slow down of BERT, and that the weight of the many samples in the AG News dataset may simply be harder to handle and require more resources. It is also possible that the implementations have not been optimized the same way.

As for the slow down of BERT, ALBERT's speed also evolve in a non linear fashion for the IMDB dataset. Since ALBERT is the only non linear model for the IMDB dataset, we assume that these results

cannot be entirely exploited and are caused by the environment. Moreover, ALBERT behave linearly for the AG News dataset, which further hints that the results for IMDB must be used very carefully.

ALBERT becomes as fast to train as DistilBert for the length 256 with the IMDB dataset, which corresponds to the kind of training speed we expect. On the contrary, we see that ALBERT is slower than DistilBert for all length with the AG News dataset. Both models have a linear speed evolution, showing no sign of environment unpredicted change.

While ALBERT has less parameters than DistilBert, it takes longer to train, which is strange. This may be caused by the nature of the parameter reduction techniques used. The distillation of DistilBert makes it a really simple model, and thus it can be trained like a regular model. On the other hand, repeating layers heavily reduce the number of parameters, but also make them harder to train. While this technique reduces the training time compared to BERT, it is probably the reason why ALBERT is slower than DistilBert.

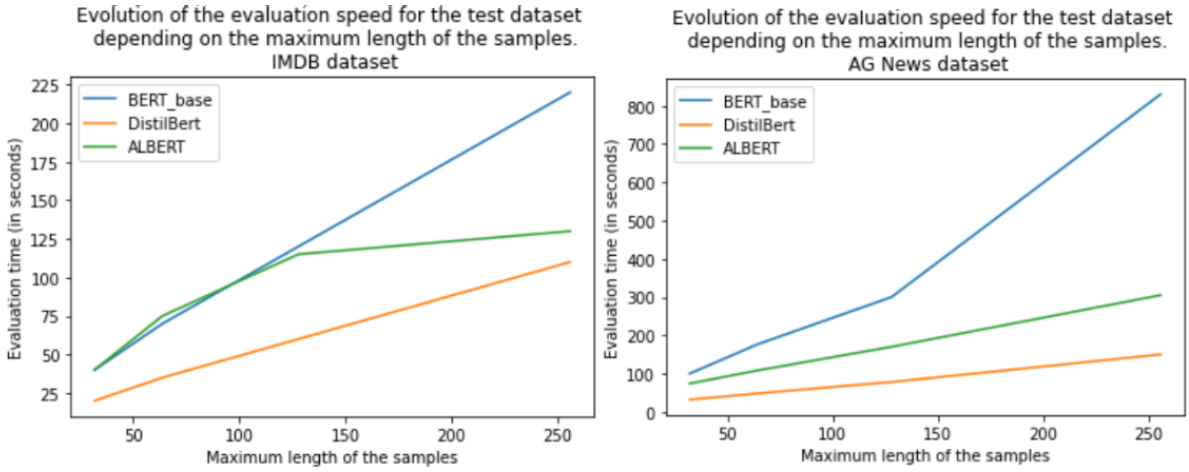

Figure 12: Relation between evaluation time and length of the samples.

The Fig 12 shows the evaluation speeds of each model depending on the length of the samples.

The results are very similar to the ones in Fig 11 we just discussed. It is thus not necessary to dive into them, as it would not bring any further information.

We can see the same non linearity as mentioned before. The overall results are also the same, with BERT being the slowest model, followed by ALBERT and DistilBert being the fastest. The scale of the gap between the different models is also identical to the one reported before.

These measures show mainly one thing: The reduction of the length of the samples also impact the evaluation speed of the models, and with the same scale as it affects the training speed.

Although rarely mentioned in the literature, the length of the samples has a major impact on the models and especially on their speed. Indeed, our results showed that it is possible to truncate the

samples approximately to the average length of the dataset without loosing too much accuracy, but while gaining a huge speed boost. The length of the sample thus has a huge impact on the overall performance of a model for text classification.

4.3 Distribution of the accuracy among the samples

As we just saw, the length of the samples has a great impact on the speed of a model, and a minimal one on the accuracy. In this subsection, we aim to dive deeper into the characterization of this hyperparameter.

Until now, the impact of the length of the sequence was measured using the same length for training and evaluation, meaning a model was trained with samples of length L_1 and its performance evaluated using samples of length L_2 with $L_1 = L_2$.

We argue that this may be limiting the performance of our models. It might be possible to train a model on a given length L_1, and to use it to predict labels for sequences of another length L_2 with $L_1 \neq L_2$. For instance, imagine training with samples truncated/padded to 256, and predicting the labels of samples truncated to 64. The training may be slower, but our previous results showed longer sequences also increased the accuracy. This knowledge of the network may enable it to better predict labels for shorter samples, and thus we could truncate the test samples to 64 to improve the evaluation speed while gaining accuracy thanks to the heavier training. To summarize, we may improve both accuracy and speed by leveraging the training length and the evaluation length separately.

Moreover, studying how well each model perform depending on the "true" length of the samples could provide more granularity regarding the performance of each model.

The results from Fig 13 to Fig 18 show how well different models perform depending on the length of the evaluation samples.

Before discussing those image, a word on how they were obtained. We took each model and trained them with a certain length: either 64 or 256. Then, the samples of the test dataset, which were not truncated nor padded, are fed to the model. For each length, we report the average accuracy obtained. Since we add a lot of granularity by separating each length, our plots are very dense (1 point per possible length). There also are many length for which there is only one sample, hence why some dots are equal to 1 or 0. When there are several samples for a given length, the average accuracy is used.

The first thing we can notice is how similar the results are regardless of which training length was used.

Indeed, there is only minimal differences between the models trained with sequences of 94 words and the models trained with sequences of 256 words. This confirms our results in the previous subsection: the length of the samples only has a minimal impact on the accuracy of each model.

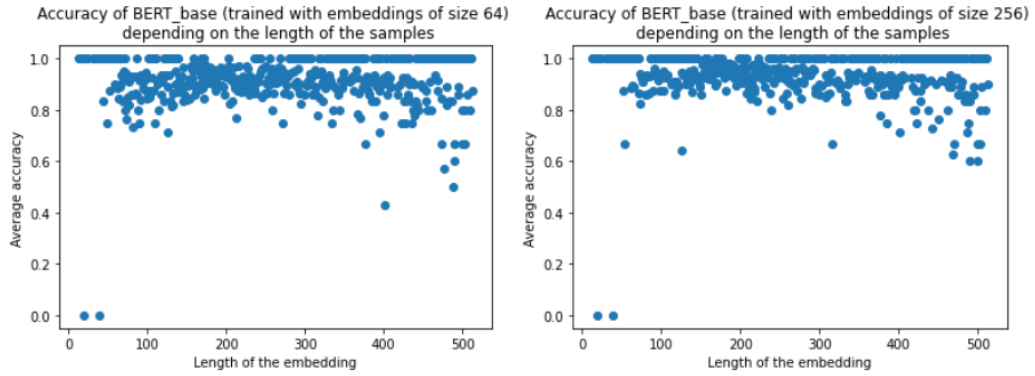

Figure 13: Detailed accuracy of BERT_base for the IMDB dataset.

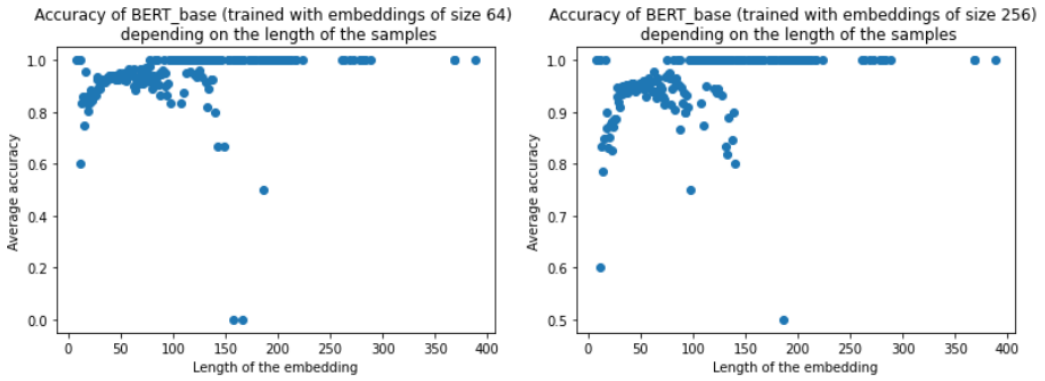

Figure 14: Detailed accuracy of BERT_base for the AG News dataset.

The results in Fig 13 and Fig 14 show little variation when using training samples of size 64 or 256. Without surprise, all the dots are mostly near the highest score (accuracy of 1).

The scores for the IMDB dataset do show a minimal difference between both training length. Indeed, the plot for the model trained with a length of 256 appears more compact than for a length of 64. It seems samples being around 256 words long are more compact, but it is also the case for those being shorter or longer.

Overall, it seems training with a higher length (256) improved the accuracy of the model for all sizes of samples when compared to the model trained with a shorter length (64). This contradicts our first intuition, which was that training a model with a certain length would mainly improve its accuracy on samples of such length.

The results for the AG News dataset provide more insight regarding this problem. It appears training on longer sequences actually decreased the accuracy of the model for short samples. The plot is indeed less compact with a training length of 256 than with 64.

For really short samples (0-50), training with shorter samples increases the accuracy. This may not be visible on the IMDB dataset since it does not have a lot of short samples. The AG News also shows that

samples of size 150-200 are slightly better labeled with a higher training length. This hints that our first intuition was correct and that it is better to choose a training length corresponding to the real length of the samples. It also shows that samples can still be truncated with minimal impact.

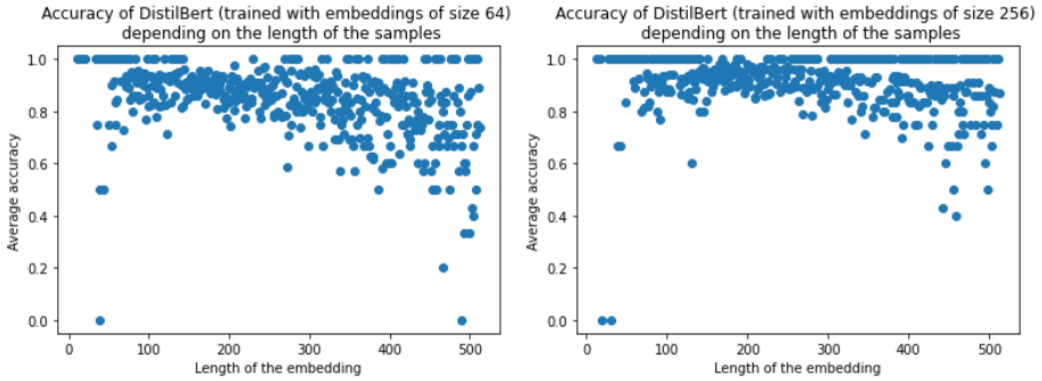

Figure 15: Detailed accuracy of DistilBert for the IMDB dataset.

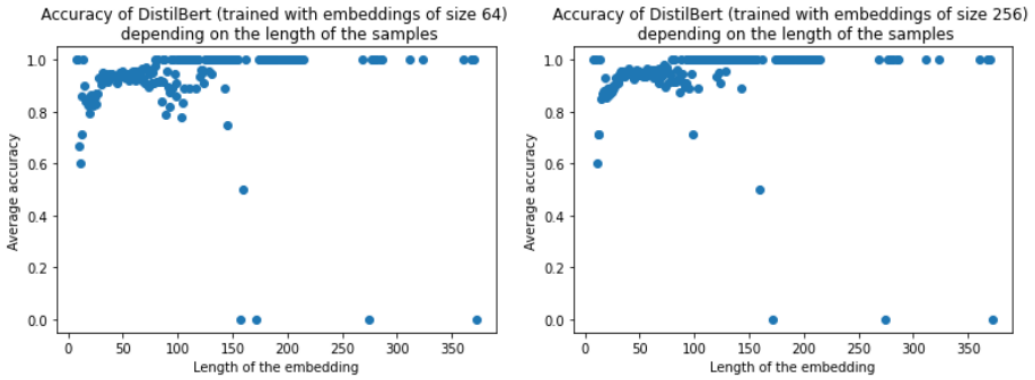

Figure 16: Detailed accuracy of DistilBert for the AG News dataset.

DistilBert behave similarly to BERT for the IMDB dataset. We can see that an increased training length improves the performance for all samples. The difference between both training length is easier to notice with DistilBert for the IMDB dataset.

DistilBert is clearly less accurate than BERT, which confirms our previous results. The difference of accuracy is especially marked for longer sequences, which BERT handles better. This difference can be explained by the complexity of both models. The heaviness of BERT may give it the upper hand for longer sequences, even with shorter training data.

Regarding AG News, the accuracy of DistilBert is increased when trained with samples of 256 words, even when labeling shorter samples. Unlike BERT, training DistilBert with longer sequences seems to always increase its accuracy regardless of the length of the tested samples.

Once again, the difference is minimal between the models.

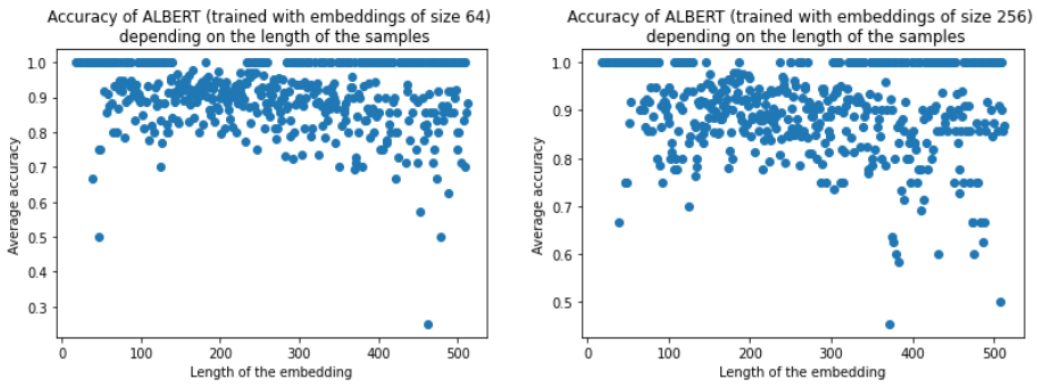

Figure 17: Detailed accuracy of ALBERT for the IMDB dataset.

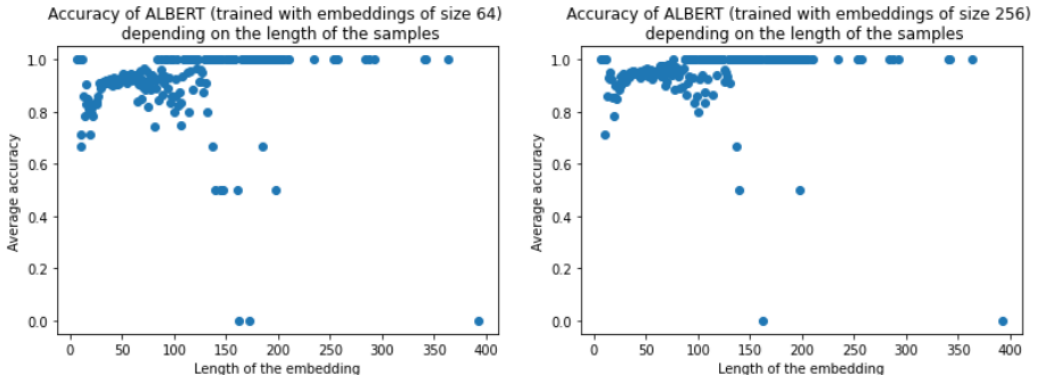

Figure 18: Detailed accuracy of ALBERT for the AG News dataset.

ALBERT is, once more, the black sheep of our models.

Unlike DistilBert and BERT, its accuracy decreases if trained with longer samples with the IMDB dataset. ALBERT obtains more compact results with a shorter training length. On the opposite, a longer training length makes the dot more compact with AG News dataset.

It is worth noting that a model can have less compact dots and yet be more accurate, which is the case here as ALBERT with $L = 256$ is more accurate than with $L = 64$ with the IMDB dataset. This is because our plots do not represent the quantity of samples, but simply their length. A plot with lots of scattered dots simply shows more volatility among the accuracy, not necessarily more accuracy.

The results for ALBERT are hard, if not impossible to explain. However, it does show that there is no "easy" relation between the training length and the capacity of a model to perform well on other sequence's length.

Overall, the results show that BERT performs better when trained on samples having a length representative of the length of all samples in a given dataset.

DistilBert on the other hand, gets better results when trained with longer sequences, even if the

training length is significantly above the average length of the samples. However, the gain of accuracy is minimal when using a higher training length than the average length of the dataset.

The complexity of BERT seems to make it unable to handle abnormal data (data that are too short/long compared to the training length) as well as DistilBert does.

The most important information shown in this subsection is that the training length has a complex relation with the capacity to process samples of various length. There are probably many other factors contributing to it, like the complexity of the data.

To summarize, the only thing we know for sure is that we can train a model with a certain training length L_1 and its performance on samples of length L_2 with $L_1 \neq L_2$ is non linear and hard to predict (if not impossible). For instance, training DistilBert on long samples makes it better even with short test samples, while this is not the case for BERT. Each model must be tested for a given dataset to choose the best training length.

On top of that, the results confirm that the reduction of the training length have a minimal impact on the accuracy. We can for instance train some models on truncated samples, and they will conserve most of their accuracy even if tested on longer samples.

4.4 Comparison and discussion

The three models used during this project all share a common architecture core: the transformer. However, they also have very distinct particularities, which are responsible for their respective strengths and weaknesses. In this section, we summarize the findings of the project and discuss the results.

The models BERT, DistilBert and ALBERT all get a good accuracy score on both datasets. They all show clear discrimination capabilities for sentiment analysis and topic classification.

Moreover, they can efficiently process text samples of various length. All networks were able to handle the data of both datasets, with results showing a clear and stable trend. This trend proves that the models are stable regardless of the length of the samples.

The stability and general good performance of the models can be explained by their shared common architecture. They are all composed of stacked encoders, but also have some important differences like the number of encoder or the use of parameter reduction techniques. Our results demonstrate the power of encoders for text classification.

More specifically, we can argue that the best performance of our model is due to the extensive pre training they endured. Indeed, weights used to initialize each network were obtained after pre training them on more than 3 billions words, which gave them a general understanding of the english language even before we performed fine tuning. The main reason behind this guess is that massive models such as BERT usually need a lot of sample to train on. In our case, the training sets would probably not have

been sufficient to train BERT and its 110M parameters from scratch. While lighter models like DistilBert and ALBERT might have gotten better results than BERT is trained from scratch, the pre training still gave them general understanding which definitely helped their accuracy. Thus, pre training probably is the main factor of the good performance of our models.

Another interesting point about pre training is that is really lowered the number of data necessary for training. When we fine tuned our models, all of the best validation accuracy were obtained using 1 or 2 training epochs. Considering the numbers of parameter in the models, the fact that they needed only one epoch to train shows how efficient and useful the heavy pre training is.

Without much surprise, BERT is the model with the best accuracy, all possible training length considered. This is no surprise for several reasons. As mentioned before, BERT is the most massive model used for this project, and therefore has a better abstraction capacity than its variants.

In addition, said variants were explicitly made to be lighter and faster than BERT. They use parameter reduction techniques, which are not helpful to increase accuracy in any way. In their respective papers, DistilBert [14] and ALBERT [15] both report having lower results than BERT. BERT being the most accurate is completely normal and expected.

What was not expected was to see DistilBert surpass BERT for some specific length of samples. More generally, it is amazing to see how close DistilBert was to BERT in terms of accuracy. It even exceeded our expectations, by getting more than 97% of the accuracy of BERT.

This shows that, despite its simplicity, the distillation procedure used by DistilBert is very efficient. It also certainly shows that our datasets are quite simple, especially AG News, for which there was only a 3% gap between the worst and best model (for all the different length of samples). It shows that the data are relatively easy to discriminate. The performance of DistilBert may be magnified by such dataset, making it appear slightly better than it actually is.

Nonetheless, DistilBert proves to have copied BERT well and manages to get a quasi non-existent gap between its accuracy and BERT's. This statement holds, even for sentiment analysis.

Those similar results beg the question: Is BERT really necessary? Is this massive and complex model really required for common classification tasks. It seems not, as a lighter model can be as good as BERT for topic classification and *almost* as good as BERT for sentiment analysis.

One of the most interesting information obtained through this project is that DistilBert is more accurate than ALBERT. It has a better accuracy in *all* situations, regardless of the length of the samples and of the dataset used. There is an almost constant gap between the accuracy of each model. On the IMDB dataset, DistilBert has 3% more accuracy than ALBERT, and 1% more than ALBERT on the AG

News dataset. We can see in Fig 10 that their curves are basically shifted identical copies of each other, with a shift of 0.03 accuracy for IMDB and 0.01 accuracy for AG News.

We can attribute this accuracy difference to the lower number of parameters of ALBERT. ALBERT is lighter than all other models, which may limit its abstraction and language understanding abilities. However, BERT and DistilBert have similar results despite a huge difference in the quantity of parameters. It is likely that, above a certain threshold number of parameters, adding more parameters does not further increase the accuracy. In the case of BERT and DistilBert, they may both be above said threshold, while ALBERT is below. BERT and ALBERT may be too complex for the tasks of this project, but ALBERT may actually not be complex enough, hence why it has a significantly lower accuracy.

Overall, the embedding factorization and repeating layers still enable ALBERT to obtain a great accuracy score, but they appear less efficient than the simple distillation used by DistilBert. We cannot definitely promote a parameter reduction technique as being better, but distillation has the upper hand for the moment. The repeating layers heavily reduce the complexity (parameters) of ALBERT, which makes it lighter than DistilBert in theory.

Accuracy is not the only important metric of our project. In fact, the reason DistilBert and ALBERT exist is because, in many cases, it is worth it to sacrifice a little bit of accuracy to increase the speed of a model. ALBERT and DistilBert were made to be less accurate but faster than BERT.

Thus, it is no surprise to see that BERT is the slowest model during both training and evaluation. It is indeed slower for both datasets and for all length of samples. This is simply due to its complexity and massive architecture (110M parameters).

As expected, ALBERT is faster than BERT. For a given length of sample, ALBERT is always faster both in training and evaluation. It is twice faster to train on the IMDB dataset, and even more than that on the AG News dataset. Regarding the evaluation time, it is 1.5 times faster on IMDB and more than twice faster on AG News.

It shows that, while ALBERT definitely is less accurate than BERT, its parameter reduction techniques enable it to be way faster than BERT. The repeating layers, which are responsible for the drastic reduction in parameter number, seem to perform well, enabling ALBERT to only include 12M parameter and to train faster.

If we take into account the accuracy of ALBERT, we can see that the reduction of parameters came at the cost of accuracy. However, the number of parameter was reduced by a factor 10 while the accuracy only dropped by a few percents in comparison. The trade off benefits ALBERT.

In addition, ALBERT was also faster than BERT during evaluation. While the repeating layers greatly speed up the training process, they do not affect the evaluation time, as the data still goes through the same number of layers. The gain of evaluation time can only be explained by the other parameter reduction

technique: the embedding factorization. This reduces the time of embedding time of the samples, making their projection into hidden vectors significantly faster in theory.

It is worth noting that, for the IMDB dataset, ALBERT appears to have the same evaluation time as BERT for several different lengths, but is also faster than BERT for other lengths. This cannot be explained by the architectural differences of the models. We assume this is caused by a change in the allocation of resources, or by an optimization difference in their respective implementations.

Overall, ALBERT is less accurate than its competitors, and is significantly faster than BERT.

When comparing the two variants of BERT, we were expecting ALBERT to be faster than DistilBert. Indeed, ALBERT has less parameters to train and a lower accuracy than DistilBert. It would make sense for the least accurate model to be the fastest. To our surprise, this was not the case.

DistilBert appears to best ALBERT for all metrics. As mentioned before, DistilBert is more accurate in all cases, and Fig 11 and 12 show it also is faster in almost all situations.

The superiority of DistilBert is interesting. It shows that a model using simple parameter reduction techniques can beat a more complicated model in both speed and accuracy, while still having more parameters. It shows that applying the KISS philosophy (Keep It Simple Stupid) is relevant even in deep learning!

The repeating layers of ALBERT have no effects during evaluation and DistilBert is thus faster because the samples have to go through less layers than with ALBERT. However, ALBERT is supposed to have 6 times less parameters than DistilBert, making it theoretically faster to train. We can explain the relatively slow training process of ALBERT by its more complicated parameter reduction techniques. ALBERT has less parameters, but those parameters are expected to be repeated, hence training a parameter of ALBERT is longer than training a parameter of the simple DistilBert model.

Another interesting fact about our model is the quasi linearity of their training and evaluation times. Indeed, DistilBert, which is twice lighter than BERT, is twice faster to train and to evaluate. This can also be said of ALBERT, despite the fact that the speed results for the IMDB dataset are not reliable enough to be exploited.

Overall, DistilBert appears to surpass ALBERT for all metrics. The complicated parameter reduction techniques of ALBERT are not as efficient as the distillation, which is also much simpler. This superiority of DistilBert may actually be the reason DistilBert was not mentioned in the original article of ALBERT. Indeed, the team made no comparison of DistilBert and ALBERT, despite DistilBert being already released to the public at this time.

It is worth noting that our speed results must be taken with a grain of salt. Since the resources allocated are often unstable, we may have some non exploitable data. However, the general trend we

reported should hold, especially since most of the results correspond to our expectations. The results for the AG News dataset also show a linear evolution of the speed for all models, which implies there were no significant abnormal changes in resources allocation.

This project brought attention on a hyperparameter: the length of the samples. The impact of the length to which we pad or truncate the sequences is immense. It may actually be more important than the choice of the models themselves. Indeed, we showed it was better to choose BERT with a sequence length of 64 rather than ALBERT with a length of 256 on the AG News dataset for instance. In this example, BERT is both more accurate and faster than ALBERT, thanks to a lower sequence length.

Shorter sequences are easier and faster to process for transformers. Indeed, the attention mechanism requires to test all tokens in the sequences against each others ($O(n^2)$). Over-padding the sequences can thus seriously reduce the speed of a model. While the paddings will quickly be recognized as such, the transformers still have to process them. Moreover, a transformer outputs a sequence as big as the one it got as input. Even if the paddings are 0, which are easy to compute, they still increases the computation time and resources consumption.

Our results showed no general truth regarding the "best" length. It must be found by testing different values for given dataset. However, we found some trends to help make a better choice. BERT performed better when given samples that were little truncated/padded. Its accuracy dropped when using samples that were too truncated or with too much padding. DistilBert and ALBERT seemed to also loose accuracy when the samples were too truncated. However, their accuracy kept increasing with over padded samples and they reached a stable plateau.

Over padded sequences can be fed to DistilBert and ALBERT, but not BERT. Depending on the necessity of a project, the length of the sequence is an interesting hyperparameter to leverage. For instance, it could be interesting to train DistilBert on longer sequences and to use it to evaluate shorter sequences, thus increasing the training time but also the accuracy, while keeping the evaluation time low.

Generally speaking, truncating samples have little effect on the accuracy of the models. It is possible to truncate samples for both training and evaluation to improve the speed of a pipeline, while conserving a high enough accuracy. Since the accuracy curve has a logarithmic shape, we can choose the training length corresponding to the beginning of the plateau to ensure the best speed/accuracy ratio.

Once again, there is no consensus regarding the best length, which depends on a model, a given dataset and the expectations toward the model: How important is speed, how much accuracy is needed... It appears that the length with the optimal accuracy/speed ratio can be obtained by choosing the average length of the sequences in the dataset. It is a safe bet, since it minimizes padding and truncation, which both reduce the quality of the data (either by removing information or by polluting it with irrelevant tokens). Our results also show that the accuracy curves seem to reach their plateau at the length corre-

sponding to the average sequence of our datasets.

To summarize, our project shows that BERT is massive, but may be unnecessary for many simple tasks. Its variants ALBERT or DistilBert are powerful enough to perform many tasks while being significantly faster thanks to their parameter reduction techniques.

Among them, the simple distillation method appears to outperform the more complicated repeating layers, making DistilBert both more accurate and faster than ALBERT.

We also showed the importance of the length of the sequence, as it can be tuned to heavily increase the speed of a model while conserving a maximum of accuracy.

5 Conclusion

My project for the course SYS843 of the ETS was the opportunity for me to learn about sequence processing. More specifically, my project was focused on text classification and the state of the art models BERT [9], as well as its variants DistilBert [14] and ALBERT [15].

Those models are composed of encoders and use the attention mechanism to process text sequences. They are considered to be the best models for such tasks as of today. The models all show great results on our datasets.

As expected, our results show that BERT is the most accurate model thanks to its complexity and massive architecture of more than 110M parameters. However, this heavy architecture also makes it the slowest of our models.

Indeed, its variants DistilBert and ALBERT use parameter reduction techniques to greatly increase their speed while retaining as much accuracy as possible.

The distillation technique of DistilBert, as well as the repeating layers and embedding factorization of ALBERT proved efficient. The variants were significantly faster than BERT, while maintaining a high accuracy.

Although distillation may look simple compared to the parameter reduction techniques of ALBERT, our results show that DistilBert always outperforms ALBERT, while also being significantly faster despite having more parameters. DistilBert is also nearly as accurate as BERT, while being twice as fast.

BERT remains the most accurate, followed by DistilBert and finally ALBERT, while DistilBert is the fastest, followed by ALBERT and finally BERT. Overall, the massive architecture of BERT is unnecessary for many tasks like topic classification and even sentiment analysis. DistilBert can perform almost as well and twice as fast !

Our project also demonstrated the importance of choosing the right length for our sequences. Indeed, the length can be leveraged to increase the speed or improve the accuracy of certain models. The results showed that it was possible to truncate the samples (to a certain extent) in order to significantly increase the speed of a model, while conserving most of the accuracy. It is possible to train a model using truncated samples, and to use it to predict the labels of longer sentences for instance, which shows that the semantic information does not require all words.

It could be interesting, in a future project, to further study the impact of the length of the samples. In addition, we could try to further improve the variants of BERT, by using distillation to train ALBERT, or the factorization embedding on DistilBert. Both model have their strengths, and merging some of their particularities could create an even faster and more accurate text processing model.

References

[1] Yoon Kim. "Convolutional Neural Networks for Sentence Classification". New York University, 2014.

[2] Alexis Conneau, Holger Schwenk, Loïc Barrault, Yann Lecun. "Very Deep Convolutional Networks for Text Classification". Facebook AI Research, LIUM, University of Le Mans, France, 2017.

[3] Ye Zhang, Byron C. Wallace. "A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification". University of Texas at Austin, 2016.

[4] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. "Efficient Estimation of Word Representations in Vector Space". Google Inc., Mountain View, CA, USA, 2013.

[5] Mike Schuster, Kaisuke Nakajima. "Japanese and Korean Voice Search". International Conference on Acoustics, Speech and Signal Processing, IEEE (2012), pp. 5149-5152.

[6] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. "Learning Word Vectors for Sentiment Analysis". The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011), 2011.

[7] Antonio Gulli. "http://groups.di.unipi.it/gulli/AG_corpus_of_news_articles.html". 2005.

[8] Ji Young Lee and Franck Dernoncourt. "Sequential Short-Text Classification with Recurrent and Convolutional Neural Networks". MIT, NAACL 2016.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". Google AI Language, 2019.

[10] Pengfei Liu, Xipeng Qiu, Xuanjing Huang. "Recurrent Neural Network for Text Classification with Multi-Task Learning". Shanghai Key Laboratory of Intelligent Information Processing, Fudan University, 2016.

[11] Jeffrey Pennington, Richard Socher, Christopher D. Manning. "GloVe: Global Vectors for Word Representation". Computer Science Department, Stanford University, Stanford, CA, 2014.

[12] Alex Sherstinsky. "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network". "Physica D: Nonlinear Phenomena", Volume 404, 2020.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. "Attention is all you need". In Advances in Neural Information Processing Systems, pages 6000–6010.

[14] Victor SANH, Lysandre DEBUT, Julien CHAUMOND, Thomas WOLF."DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". Hugging Face, 2019.

[15] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, Radu Soricut. "ALBERT: A LITE BERT FOR SELF-SUPERVISED LEARNING OF LANGUAGE REPRESENTATIONS". Google Research, Toyota Technological Institute at Chicago, 2020.