Universitat Politècnica de València
Dept. of Computer Systems and Computation

Neural news classifier from pre-trained models

Author: Vázquez Barrera, Adrián
Advisor: Casacuberta Nolla, Francisco
2021-2022

Master's Thesis

Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging.

Table of contents

List of Figures

List of Tables

Abstract

It is proposed to study the pre-trained linguistic models available in PyTorch, in order to fine-tune them and improve their baseline accuracy for the task of news classification.

This work reviews the current state of the art of text classification as well as current techniques and classical methods, with a special focus on transformer-based models. It will also address the problem of corpus preprocessing for this kind of task. The study will take into consideration metrics such as training cost, accuracy, precision, recall and F1-Score in order to be able to compare them.

It is essential to ensure that the results obtained in this study are in line with those achieved by other researchers using the same corpora, therefore, despite not having used the same data partitioning or the same techniques, it has been considered appropriate to mention them later on in this study.

The model is intended to be usable by end users, so the best implementation has been released for production through a web interface.

Resumen

Se propone estudiar los modelos lingüísticos preentrenados disponibles en PyTorch, con el fin de afinarlos y mejorar su precisión de base para la tarea de clasificación de noticias.

Este trabajo revisa el estado actual de la clasificación de textos, así como las técnicas actuales y los métodos clásicos, con especial énfasis en los modelos basados en Transformers. También se abordará el problema del preprocesamiento del corpus para este tipo de tareas. El estudio tendrá en cuenta métricas como el coste de entrenamiento, la fiabilidad, la precisión, el recall y la F1-Score para poder compararlas.

Es fundamental que los resultados arrojados en este estudio estén en consonancia con los obtenidos por otros investigadores utilizando los mismos datos, por lo que, a pesar de no haber utilizado la misma partición de datos ni las mismas técnicas, se ha considerado oportuno mencionarlos más adelante en este estudio.

Se pretende que el modelo sea utilizable por los usuarios finales, por lo que la mejor implementación se ha puesto en producción desde una interfaz web.

Resum

Es proposa estudiar els models lingüístics preentrenados disponibles en PyTorch, amb la finalitat d'afinar-los i millorar la seua precisió de base per a la tasca de classificació de notícies.

Aquest treball revisa l'estat actual de la classificació de textos, així com les tècniques actuals i els mètodes clàssics, amb especial èmfasi en els models basats en Transformers. També s'abordarà el problema del preprocessament del corpus per a aquesta mena de tasques. L'estudi tindrà en compte mètriques com el cost d'entrenament, la fiabilitat, la precisió, el recall i la F1-Score per a poder comparar-les.

És fonamental que els resultats llançats en aquest treball estiguen d'acord amb els obtinguts per altres investigadors utilitzant les mateixes dades, per la qual cosa, malgrat no haver utilitzat la mateixa partició de dades ni les mateixes tècniques, s'ha considerat oportú esmentar-los més endavant en aquest estudi.

Es pretén que el model siga utilitzable pels usuaris finals, per la qual cosa la millor implementació s'ha posat en producció des d'una interfície web.

Keywords

BERT, Classifier, Computer linguistics, Deployment, Development, Fine-tuning, GPT, Language model, Machine learning, Neural network, News, Pre-trained, PyTorch, Transformers.

Chapter I - Introduction

1.1 Problem

Text classification is a highly relevant field [1] of study in Natural Language Processing (NLP) as it is possible to classify huge amounts of text into labels in a very short time and at a very low cost compared to what would be necessary if it were carried out by humans.

By properly training a text classifier, nowadays standardised problems can be dealt with [1], such as spam detection in e-mails [9], political bias detection [23], hate speech detection and so on. The examples mentioned earlier are mostly in the field of sentiment analysis, in this study we will try to develop a news classifier that tackles the problem from three different angles.

1.1.1 User perspective

The world we live in is constantly evolving faster and faster, so quickly that it often takes time for society to assimilate new technological developments. The progressive extinction of the written press [28] and its consequent transition to the electronic format has allowed any person or group of people to start up media at almost zero cost. Nevertheless, there is a counterpart that should not be underestimated. The proliferation of media has caused an increase in competition to gain clicks or subscriptions from customers [26]. This inevitably leads to the situation we are facing today, where apparently a headline can be deliberately written to be misinterpreted, or even worse, content that is not even minimally related to the topic in vogue (such as politics) is forced into the article to achieve relevance. All of the aforementioned have resulted in a rising distrust of the media on the user side. A system that can quickly alert users to such articles could be a time-saver for them.

1.1.2 Press perspective

As mentioned in the previous section, digital media are on the rise and traditional media are aware of this. It is therefore not surprising that they want to digitise their old printed editions to their website [27]. This problem requires two different models working together, firstly a system that automatically converts the content into digital content and then another one that sorts it into a specific section of their online newspaper, helping to search better for old documents and preserve the history of which traditional newspapers are part.

1.1.3 News aggregation perspective

News aggregators such as Flipboard[1] or Feedly[2] make sure to crawl the web and display the latest news to users. For these kinds of platforms, it is vital to classify them correctly to fit the customer's preferences, however, this is not always possible, as manual news tagging standards are barely followed. Models such as those proposed in this paper can be very useful for this kind of service.

1.2 Proposed solution

There are several approaches available to address the text classification problem. While it is possible to try to solve the task with classical methods, the use of these has been completely discarded.

Classical methods are ineffective in establishing strong relationships between words in a text [1]. Even those methods that are able to set them up inevitably face the problem of having to apply smoothing techniques, because there are words that have not been seen during training.

[1] https://flipboard.com

[2] https://feedly.com

It is also possible for a word that, despite being recorded in the model, there is no probability associated with a given context [9]. There is even a scenario where the sentence to be evaluated could be so long that the probability associated with it is negligible.

For all of the above reasons, the use of neural models is necessary. There are various topologies of neural networks that can be used for the task of text classification, and these will be discussed in more detail later on in this document.

In addition, we will use pre-trained language models (such as BERT [11] or GPT-2 [16]) to speed up training and propose custom solutions based on the best-performing one.

In this study, we will be looking at pre-trained models available in PyTorch to fine-tune them for the task of news classification. We will be considering various metrics such as training cost, accuracy and model performance in order to determine the best model for this task. The aim is to create a model that is usable by end-users, so the best implementation will be released for production.

1.3 Objectives

The purpose of this tool is to enable users to identify the main topic of an article without much effort, thus avoiding the need to read the full text. This is especially useful for customers who are short on time or who want to quickly get an overview of the main topics being discussed on a certain website or blog. By using this tool, users can save time and discard articles containing irrelevant information quickly and easily.

Therefore, the project's objectives include the following:

- Research and enumerate those state-of-the-art methods that can be used to accomplish the news classification task.
- Search for labelled dataset(s) that allow training proposed models.
- Pre-process the data and carry out a thorough evaluation of the hyper-parameters that should be taken into consideration.
- Test all neural models (pre-trained mostly) that may be able to solve this task by applying a fine-tuning technique. Considering time constraints.
- Perform enhancements on the best-performing model according to its behaviour on the pre-labelled dataset.
- Deploy the best approach so that it can be used in production.
- Draw conclusions from the experiments carried out.

1.4 Document breakdown

The document is structured in eight principal items: state of the art (Chapter I), the experimental framework (Chapter II), the environment in which the experiments will be performed (Chapter III), it is experimentation on the models (Chapter IV), a summary of the obtained results (Chapter V), a brief description of the model deployment (Chapter VI), the conclusions of the study (Chapter VII), and possible future works that may be derived from this project (Chapter VIII).

In the state-of-the-art section, there will be an overview of the text classification techniques that have been applied so far. Some classical methods will be briefly described, but there will be a particular emphasis on neural models, especially those based on transformers.

The experimental framework section will describe in detail the data used to carry out the required task, as well as the necessary pre-processing of the data and the metrics to be used. Then, in the environment section, it will be described which tools, environments and/or libraries have been used to perform the experiments.

In chapter IV, there will be a detailed review of the results obtained by each model using the metrics presented in the experimental framework. Then, in chapter V, the results obtained will be shown globally in order to compare the models in the conclusions section and finally, the future work that may be derived from this project will be mentioned.

Chapter II - State of the art

2.1 Text classification

Text classification is the technique of categorising texts into labels based on their content [1]. It is one of the most relevant tasks in natural language processing (NLP), consisting of attempt detection, topic labelling, spam detection, and sentiment analysis. In this way, text classifiers can automatically categorise texts based on a predefined set of labels.

NLP, data mining and machine learning techniques are used to find and detect patterns in electronic documents automatically. The main goal of text mining is to allow users to retrieve information by using textual tools and process operations. The objective of information extraction (IE) methods is to extract accurate data from textual documents [1]. That is the initial approach, which proposes that text mining is equivalent to data mining.

The process of searching for documents that answer questions is known as information retrieval (IR) [1]. This involves the use of statistical measures and methods for the automated processing of text data and its reference to the given query. In its widest context, information retrieval covers the whole scope of information processing, from data retrieval to knowledge retrieval.

Text classification consists of classification processing steps such as preprocessing, feature extraction and the classification model for training and prediction. The training set is used to train the model and prediction is used to forecast the most plausible labelled output.

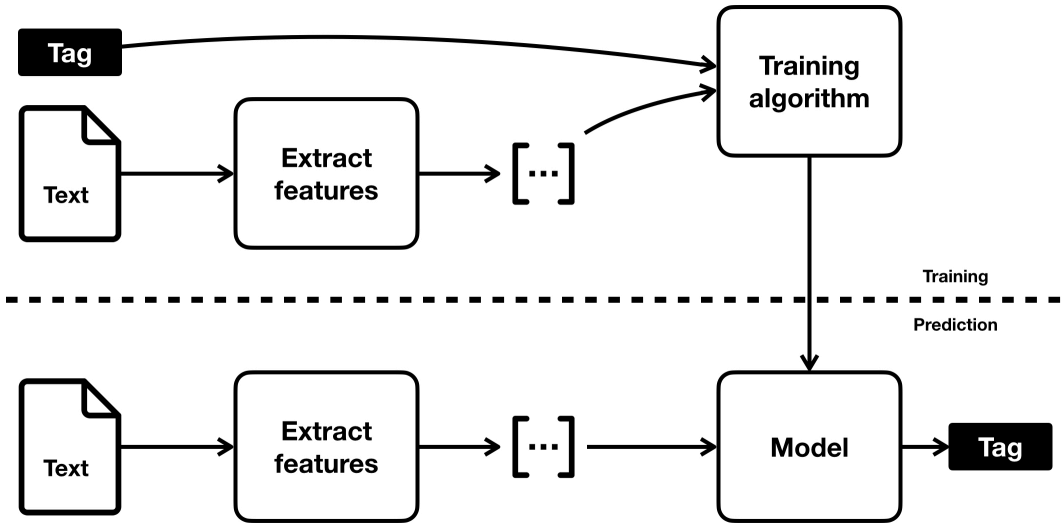

Figure 1: Common workflow scheme for text classification

2.1.1 Pre-processing

As in most cases when working with data, pre-processing is one of the most important tasks for text classification in natural language processing [1]. There are three major methods for processing. Such as:

- **Tokenisation**: involves strings that are divided into smaller tokens. It is possible to tokenise paragraphs into sentences and tokenise sentences into words.
- **Stop word removal**: this consist of eliminating common and frequent words like the, a, an etc. and removing irrelevant words.
- **Stemming**: or noise removal is the process of reducing inflected words to their word stem, base or root form.

2.1.2 Feature extraction

Feature extraction not just provides greater accuracy, it also offers higher probability and saves computational time [7]. It has different properties such as a bag of words, embedding and extraction of features from the text which are converted into different vectors. The vectors are counted in the form of TF-IDF vectors [6], namely term frequency (TF) which means the number of times a term appears in a document compared to the total number of words in the document and inverse document frequency (IDF) reflecting the proportion of documents in the corpus that contains that term.

Term frequency:

$$TF(t,d) = \frac{f(t,d)}{N}$$

Being

t a term

d a document

N total number of terms in a document

$f(t,d)$ number of times a term appears in a document

Inverse Document Frequency:

$$IDF(t,D) = log\frac{|D|}{1 + g(t,D)}$$

Being

t a term

D the set of available documents

$|D|$ total number of documents

$g(t,D)$ number of documents in which a term appears

The TF-IDF of a term is calculated by multiplying TF and IDF scores.

$$TF\text{-}IDF(t, d, D) = TF(t, d) \cdot IDF(t, D)$$

Being

t a term

d a document

D the set of available documents

$TF(t, d)$ Term Frequency

$IDF(t, D)$ Inverse Document Frequency

2.1.3 Text classification algorithms

As we can see, certain types of data can only be understood as a sequence. Unlike other networks, such as those implemented for an image recognition system, where examples are handled by a single forward. Due to their unstructured nature, texts are computationally hard to process. Despite this, they can be a significant information source.

However, thanks to advances in natural language processing (NLP), implementing this kind of task has become easier to carry out. These machine learning algorithms can be categorised under three possible types [1]:

- **Supervised**: Naive Bayes, Support Vector Machines (SVM), Neural Networks, Convolutional Neural Networks and Recurrent Neuronal Networks.

- **Unsupervised**: K-Means, Clustering, Principal Components Analysis (PCA) and Association Rule Learning.

- **Semi-supervised**: Self-training, Semi-Supervised Vector Machines (S3VM) and Multi-view pseudo-labelling.

As it can be observed, models based on Transformers have not been categorised in any of the three categories above, this is because the boundaries between them are blurred when using this

technique. Since models such as BERT [11] or GPT-2 [16] are usually pre-trained with large data sets (usually in an unsupervised way) and then transfer learning to a task using supervised samples [18].

Even using the two examples above we can observe remarkable differences: GPT-2 was pre-trained in a semi-supervised way using a very large English corpus, while BERT was trained using two unsupervised tasks: Masked-LM and Next Sentence Prediction.

Some of the techniques mentioned above are detailed in more detail later in this document. Such as:

- Naive Bayes
- Support Vector Machines (SVM)
- Recurrent Neural Networks (RNN)
- Long Short-Term Memory (LSTM)
- Connectionist Temporal Classification (CTC)
- Transformers

This project will focus on the use of Transformers and consequently on the existing models available.

In a summary, machine learning text classification learns how to make classifications based on prior observations. By using pre-labelled examples as training data, machine learning algorithms can infer the different associations between text samples.

To train a machine learning algorithm on text data, first, the text must be converted into numerical feature vectors. This can be done using various methods depending on the type of text and the desired features. When the text is in vector form, the machine learning algorithm is then given training data consisting of pairs of feature vectors and labels. The algorithm uses this data to learn how to make classifications based on prior observations.

Different types of features can be used to represent the text, such as bag-of-words, part-of-speech tags, dependency parse trees,

etc. The features are typically chosen based on what is known about the task at hand. Once the training data has been processed and the model generated, it can be used to make predictions on unseen data. To do this, the text is first transformed into a vector of features in the same way as the training data. The classification model is then applied to the features vector to generate a prediction.

2.2 Naive Bayes

The Naive Bayes method is one of the most well-known methods and was used especially at the end of the 20th century for multinomial classification tasks, in particular in the detection of junk mail [9].

A pre-labelled dataset is needed to train the classifier. From this data, it is possible to obtain the prior probability of each label, as well as the conditional probability on each class for words in the text.

$$p(c \mid W) = p(c) \cdot \prod_{i=0}^{N} p(w_i \mid c)$$

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}(p(c \mid W))$$

Being

W a sentence

w_i word of the sentence

N sentence length

C a set of available classes

c a singles class

Notice that in order to avoid misclassifications it is necessary for all the input words must have been seen previously in training, at least once per class, otherwise, we would be providing a likelihood of zero regardless of the other words in the sequence.

One way to avoid this problem is to use some kind of smoothing method. But we would not solve the main limitation, it takes a sequence of words independent of each other, no matter what order or context they are in.

2.3 Support Vector Machines

Support vector machines (SVM) are somewhat more expensive to train than naive Bayes, but it can provide more accurate results. The main idea behind this algorithm is a hyper-plane, which separates at best two or more groups of objects. An optimal hyper-plane should be that which leaves as large a gap as possible between groups [25].

The forecast hypothesis for an SVM is defined as follows:

$$h(x_i) = sign(\sum_{j=1}^{S} a_j y_j K(x_j, x_i) + b)$$

$$K(v, v') = exp(\frac{||v - v'||^2}{2\gamma^2})$$

Being

- x_i the feature vector
- b a constant
- a_j a constant for each x_j
- y_i represents the class (-1 or 1) for each x_j
- S the number of support vectors
- $K \in (0.0, 1.0)$ the kernel function (1 being identical, 0 fully different vectors)

To perform a non-binary classification of the data on tasks with more than two classes, it will be necessary to make a hypothesis for each class, that isolates each one from the others, as shown in the picture below.

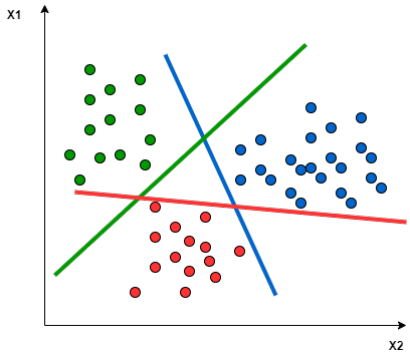

Figure 2: Multi-class SVM

Sometimes data may not be linearly separated by the initial number of dimensions, increasing the number of dimensions of data may fix this issue. In the case of text classification, it is enough to simply modify the feature extractor.

2.4 Recurrent Neural Networks

A recurrent neural network (RNN) is an artificial neural network that works with sequential data [3]. Common uses for this kind of network include speech recognition, image captioning, language translation, and natural language processing (NLP).

They are distinguished by their “memory” as they take information from prior inputs to influence the current input and output. While conventional deep neural networks consider all inputs and outputs independent of each other, the output of recurrent neural networks hangs on the sequence's prior parts.

Another feature of recurrent networks is that they share parameters across each layer of the network. Unlike feedforward networks with different weights at each node, recurrent neural networks share the same weight parameter at every layer of the network. Having said that, these weights are still adjusting in the back-propagation and gradient descent processes to facilitate learning by reinforcement.

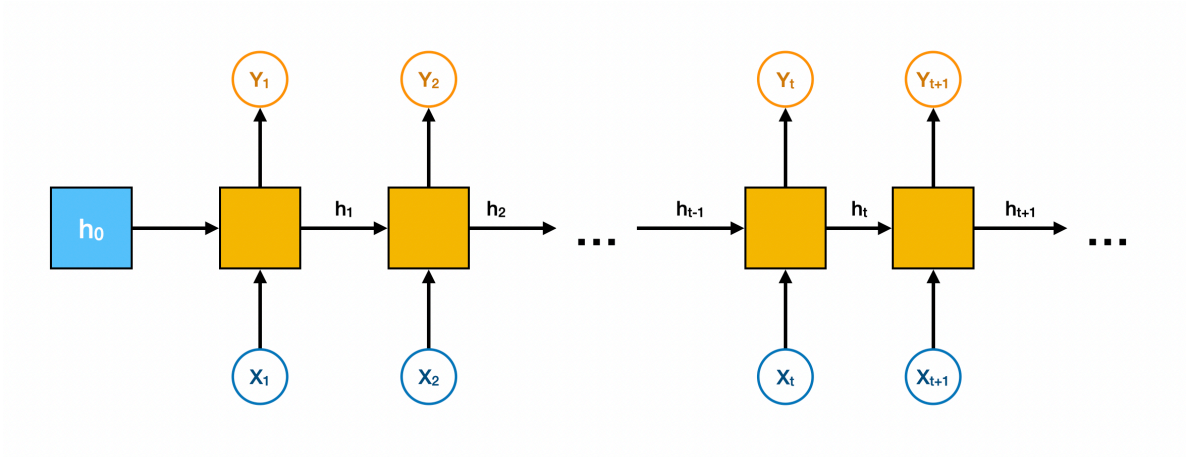

Figure 3: RNN

Being

x_t the current sequence's input

h_0 initial hidden state

h_t the current hidden state

y_t the current output

2.4.1 Back-propagation through time

Recurrent neural networks take advantage of the back-propagation through time (BPTT) algorithm to compute gradients [15], which is slightly different from traditional back-propagation as it is special for sequence data. For better understanding, we are considering this representation which matches the text classification task.

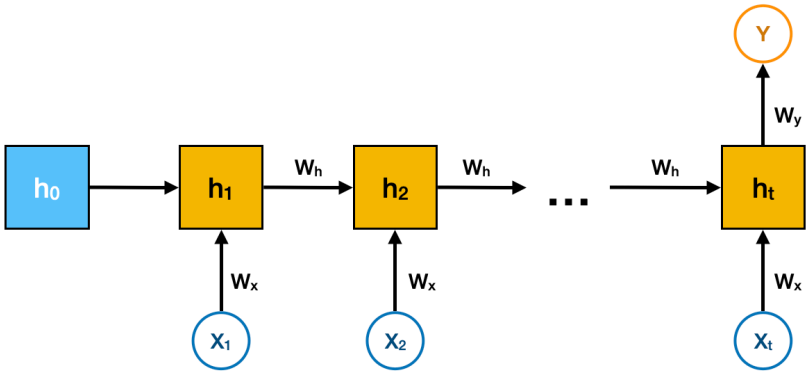

Figure 4: RNN Weights

Being

x_t the current sequence's input

h_0 initial hidden state

h_t the current hidden state

y the output

W_x the input weights matrix

W_h the hidden state weights matrix

W_y the output weights matrix

Adjust W_h:

$$\frac{\partial E_T}{\partial W_h} = \sum_{t=1}^{T} \frac{\partial E_T}{\partial Y_T} \cdot \frac{\partial Y_T}{\partial h_t} \cdot \frac{\partial h_t}{\partial W_h}$$

Adjust W_x:

$$\frac{\partial E_T}{\partial W_x} = \sum_{t=1}^{T} \frac{\partial E_T}{\partial Y_T} \cdot \frac{\partial Y_T}{\partial h_t} \cdot \frac{\partial h_t}{\partial W_x}$$

Being

E_T the error at time T

Y_T the output at time T

W_x the input weights matrix

W_h the hidden state weights matrix

h_t the hidden state at time t

There are several types of RNN such as:

- One to Many
- Many to One
- Many to Many with the same length output
- Many to Many with different length outputs
- Etc

2.4.2 One to Many

A network produces an output sequence given a single input. Its applications can be found in Music Generation and Image Captioning.

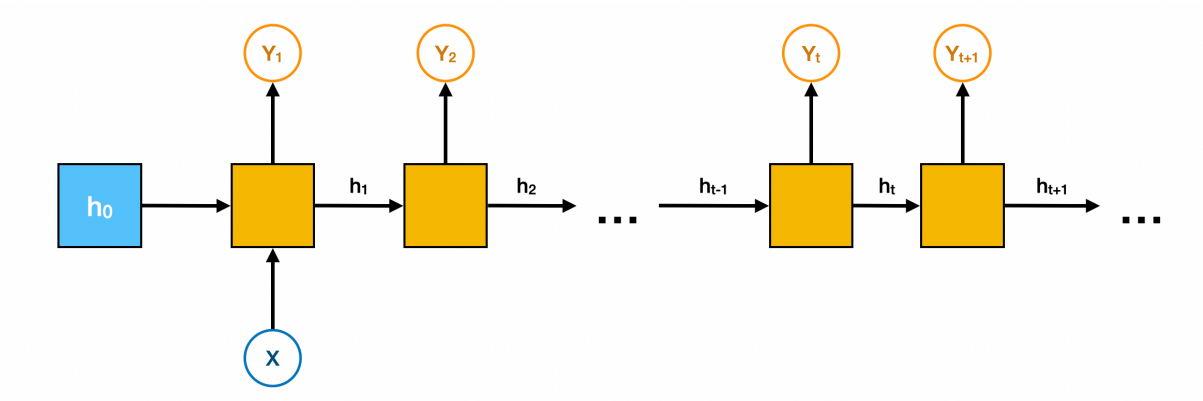

Figure 5: One to Many RNN

2.4.3 Many to One

It generates a single output given a sequence, this kind of RNN is used in text classification.

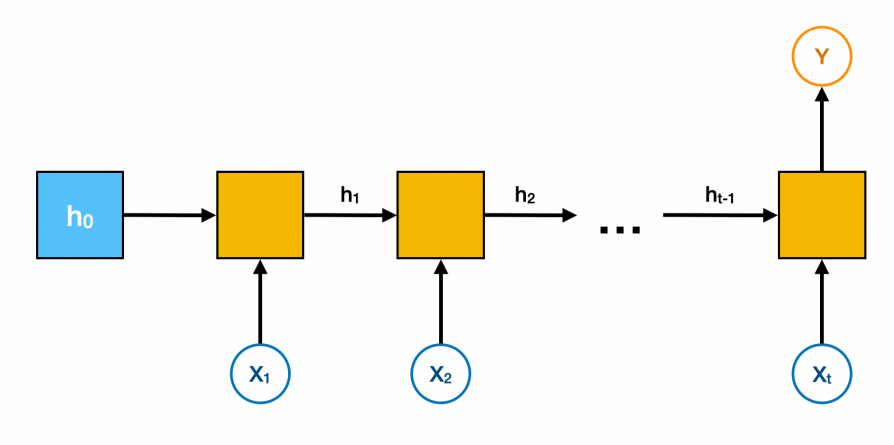

Figure 6: Many to One RNN

2.4.4 Many to Many I - Same length output

The network generates an output with the identical length of the input. A common application can be found in Name-Entity Recognition.

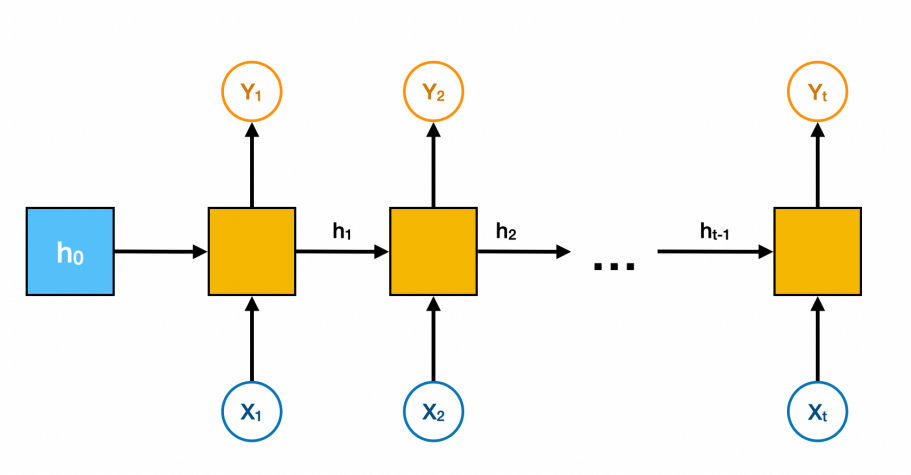

Figure 7: Many to Many I RNN

2.4.5 Many to Many II - Different length output

Mainly used in machine translation, it allows the generation of an output sequence with a different length than the input sequence.

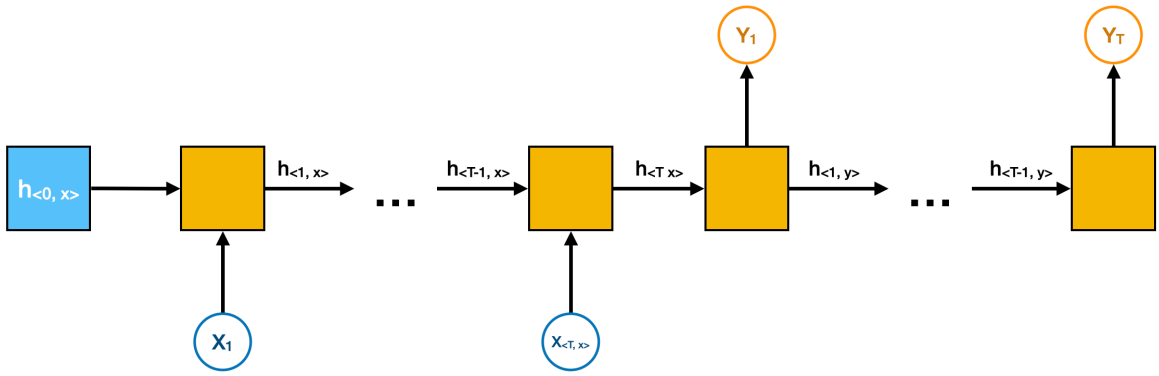

Figure 8: Many to Many II RNN

2.4.6 Bidirectional recurrent neural networks

Classical RNNs only process input texts word by word, from beginning to end. Exactly in the same way as a standard human would. However, we can also somehow intuit words or concepts later in time. Therefore, it would be even more convenient if the model could also know the future words and solve the problem more effectively. For this case, bi-directional RNNs are used.

In a bidirectional RNN [14], we consider two separate sequences. One from right to left and the other in the reverse order. The outputs are generated by concatenating the sequences at each time and generating weights accordingly. Note that both the left-to-right and right-to-left layers are never directly connected.

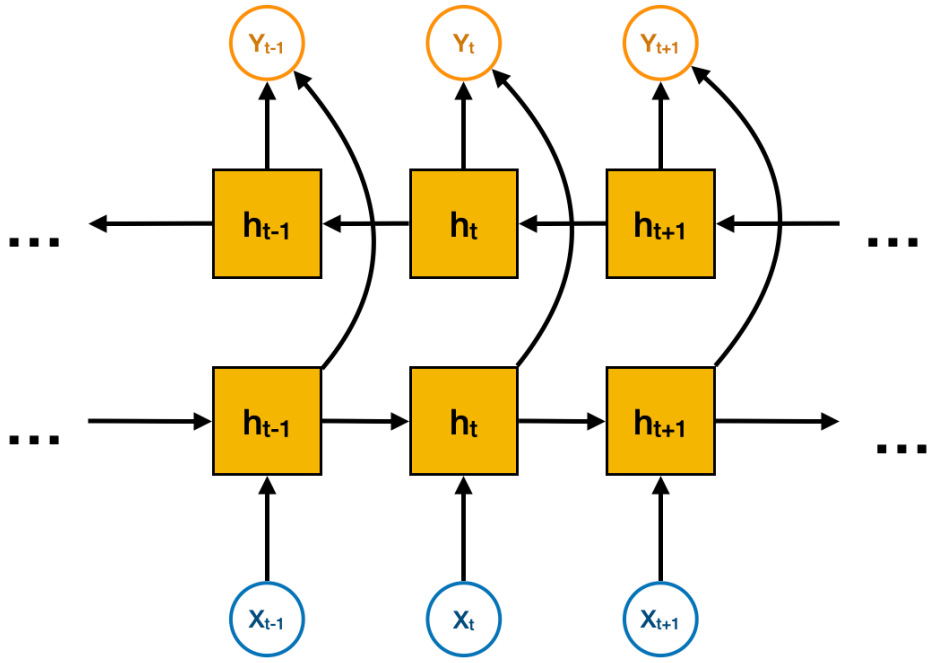

Figure 9: Bi-directional RNN

2.4.7 Connectionist Temporal Classification

Connectionist temporal classification (CTC) is a type of neural network layer for training recurrent neural networks that were designed specifically for temporal classification tasks, where the alignment between the inputs and the expected output is unknown [2][8]. It models all aspects of the sequence with a single neural network and does not require combining the network with any extra model, such as a hidden Markov model. It also does not require pre-segmented training data or additional post-processing to extract the sequence label from the network output. They are typically used for handwriting recognition or speech recognition.

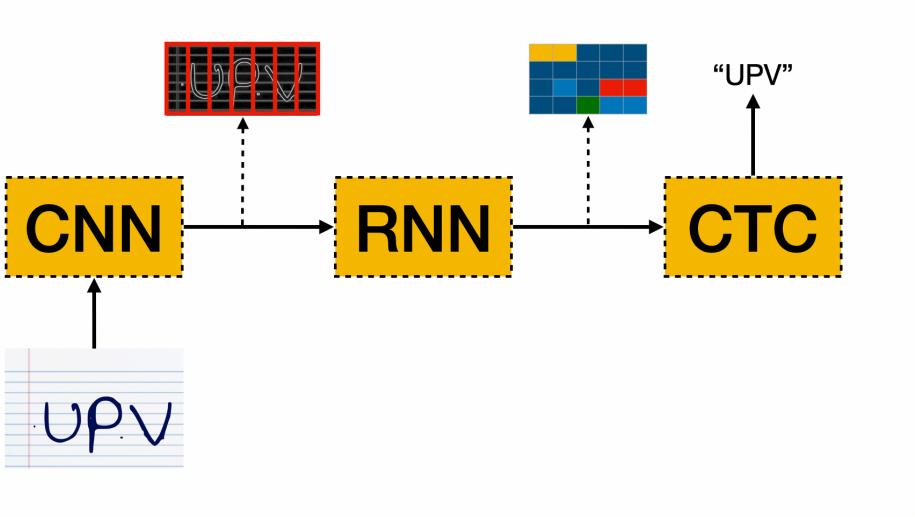

Figure 10: Example of CTC usage for handwriting recognition

2.5 Long Short-Term Memory Networks

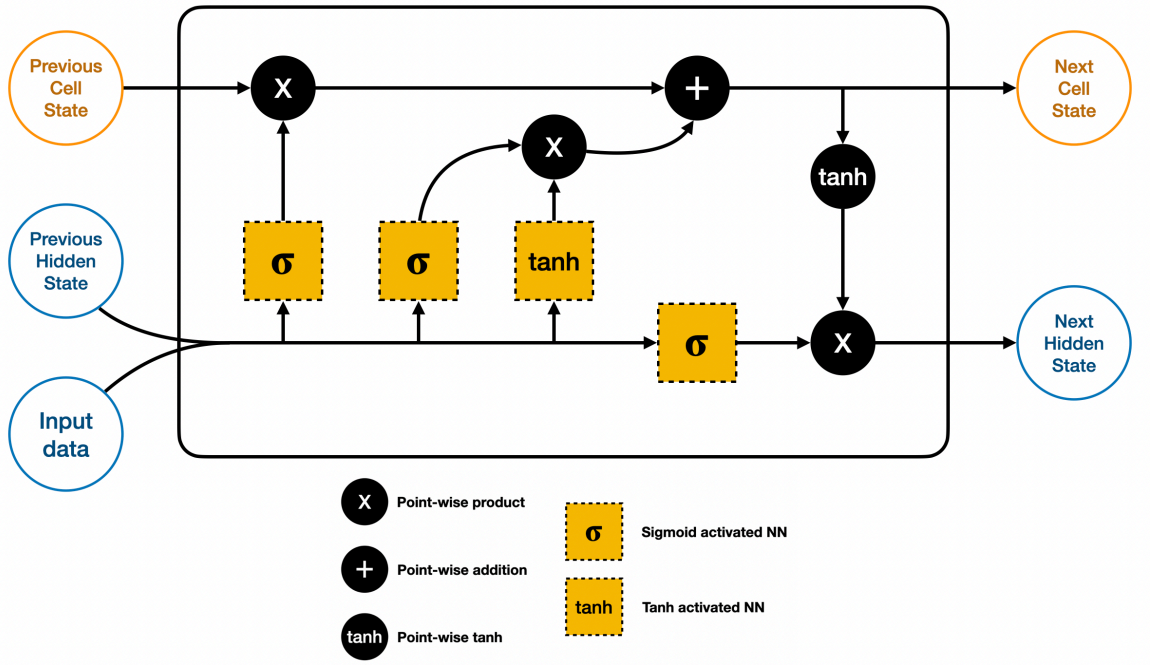

Figure 11: LSTM Overview

Long-term memory networks (LSTM) are a special kind of recurrent neural network that can learn order dependence [17]. The output of the previous step is used as input to the current step in the RNN. LSTM handles the problem of long-term dependence, where RNNs are unable to predict the words stored in long-term memory, but can make more accurate predictions based on the actual data. RNNs do not perform efficiently as the gap length increases. LSTM can preserve information for a long time by default.

Cells store information, while gates manipulate memory. There are three gateways:

- Forget gate
- Input gate
- Output gate

2.5.1 Forget gate

The forget gate is the first step in the process [10][17]. Based on the previous hidden state and the new input data, it will decide which bits of the cell state (long-term memory of the network) are useful.

To do this, we are feeding the previous hidden state and the new input data into the network. This network generates a vector where each element is in the interval [0,1] (ensured by using the sigmoid activation). In the forget gate, this network is trained to produce close to 0 when a component of the input is irrelevant and close to 1 when it is relevant. Each element of this vector can be thought of as a filter that lets in more information as the value gets closer to 1.

These output values are sent upwards and point-wise multiplied with the previous state of the cell. This point-wise multiplication means that components of the cell state which have been considered irrelevant by the forget gate, the network will be multiplied by a number close to 0 and will therefore have less influence on the following steps.

In summary, the forget gate decides which pieces of the long-term memory should be forgotten (have less weight) given the previous hidden state and the new data in the sequence.

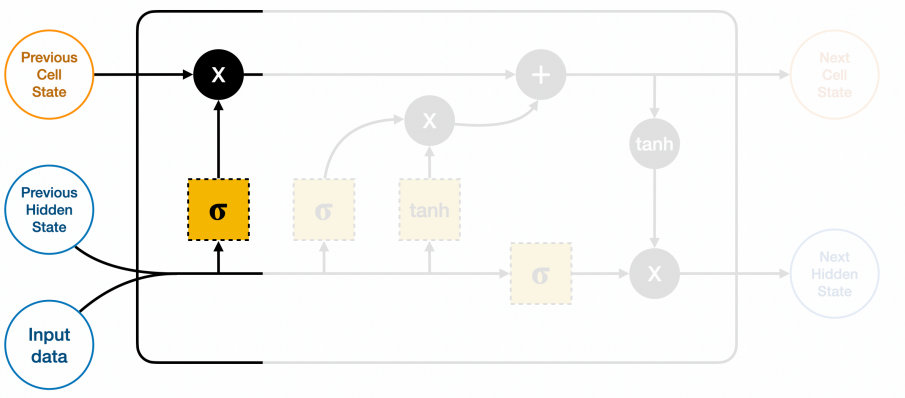

Figure 12: LSTM - Forget gate

The function which defines the forget gate is defined as:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

Being

f_t the forget gate

σ sigmoid activation function

W_f forget gate weights matrix

h_{t-1} hidden state of the previous LSTM block

x_t current input

b_f forget gate bias

2.5.2 Input gate

The next step involves the new memory network and the input gate [10][17]. Given the previous hidden state and new input data, this step determines what new information should be added to the network's long-term memory (cell state).

The new memory network and the input gate are neural networks themselves, and both take the same inputs, the previous hidden state and the input data. It should be emphasised that the inputs here are the same as for the forget gate.

The new memory network is a hyperbolic tangent-activated neural network that combines previous hidden states with new input data to deliver a new memory update vector. This vector indicates how much each long-term memory component (cell state) of the network needs to be updated given new data.

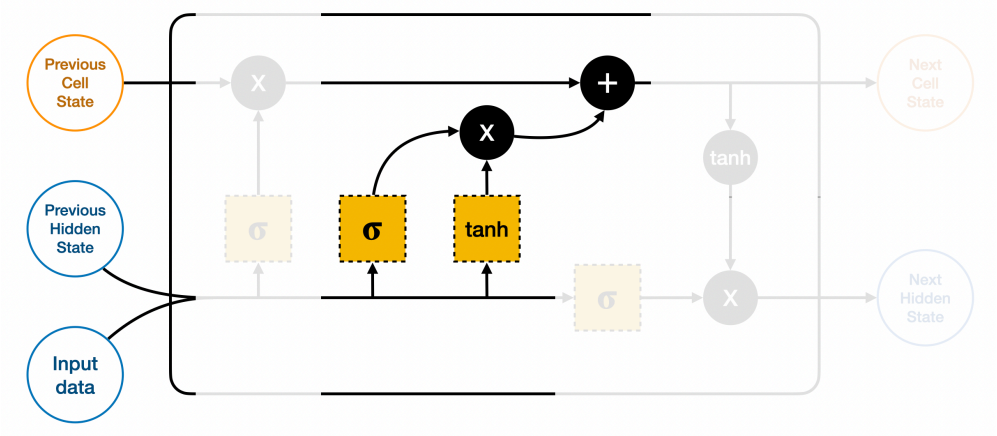

Figure 13: LSTM - Input gate

Note that we use a hyperbolic tangent here because its values lie in [-1,1] and so can be negative. The possibility of negative values here is necessary if we wish to reduce the impact of a component in the cell state.

Despite this, it doesn't actually check if the new input data is even worth remembering. Input gates play a crucial role here. The input gate is a sigmoid-activated network which acts as a filter, deciding which components of the new memory vector should be retained.

This network will output a vector of values in [0,1] allowing it to act as a filter through point-wise multiplication. An output near zero, tells us we don’t want to update that element of the cell state. Earlier outputs are multiplied point-wise. By adding the combined vector to the cell state, the long-term memory of the network is updated. The function which defines the input gate is defined as:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

Being

i_t the input gate

σ sigmoid activation function

W_i input gate weights matrix

h_{t-1} hidden state of the previous LSTM block

x_t current input

b_i input gate bias

2.5.3 Output gate

The output gate calculates the next hidden state using input data, prior hidden state and the recently updated cell state [10][17].

It is not possible to simply return the new cell state, as we would be giving back all the information accumulated so far. To prevent this from happening, a filter is simply added in the same way as in the forget gate. Therefore, a sigmoid function is used as a filter, since its range of values is between zero and one. Nevertheless, before applying the filter, the state of the cell is passed through a hyperbolic tangent to ensure a range of values in the interval [-1,1].

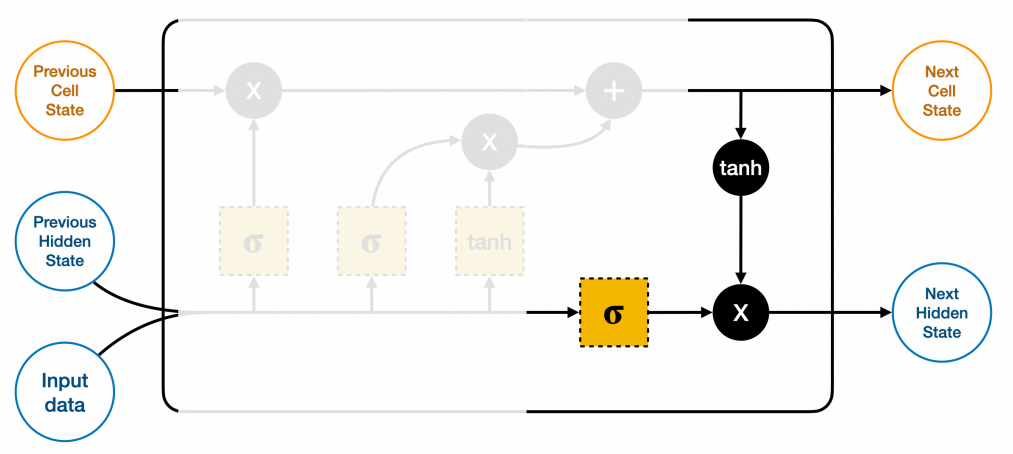

Figure 14: LSTM - Output gate

The function which defines the output gate is defined as:

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

Being

o_t the output gate

σ sigmoid activation function

W_o output gate weights matrix

h_{t-1} hidden state of the previous LSTM block

x_t current input

b_o output gate bias

The equations for the cell state are defined as:

$$\tilde{C}_t = tanh(W_c[h_{t-1}, x_t] + b_c)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

Being

- $\tilde{C}_t$ the candidate for current cell state
- C_t the current cell state
- C_{t-1} the previous cell state
- $tanh$ hyperbolic tangent activation function
- W_c cell state weights matrix
- h_{t-1} hidden state of the previous LSTM block
- x_t current input
- b_c cell state bias
- f_t the forget gate output
- i_t the input gate output

The equation for the hidden state is defined as:

$$h_t = o_t \odot tanh(C_t)$$

Being

- h_t the hidden state
- o_t the output gate output
- C_t the current cell state

2.6 Transformers

A transformer is a machine learning model that embraces the self-attention mechanism [5], differentially weighing the significance of each part of the input data.

Transformers, like recurrent neural networks (RNNs), are used for tasks like summarising and text translation since they are designed to deal with sequential data. Transformers, however, handle all of the input simultaneously, in contrast to RNNs, which process each token of a sequence one at a time. The attention mechanism provides context for any position in the input stream. For example, if the input data is a natural language sentence, the transformer does not have to process the words one by one. This allows greater parallelisation than RNNs and consequently reduces training times.

To understand how transformers work, it is necessary to explain its main parts:

- Inputs embeddings
- Positional encoding
- Self-attention mechanism
- Layer normalisation

In the following picture, the architecture of a transformer is represented the same way as in the original paper [5]. It can be split into two parts: the encoder (left) and the decoder (right). Since we are making text classification, the decoder part could be omitted.

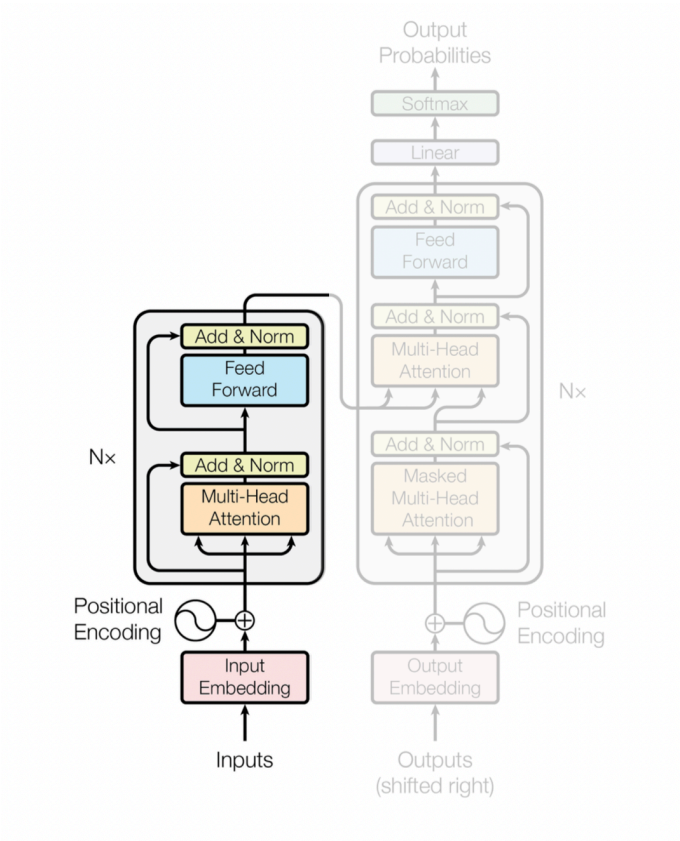

Figure 15: Transformer architecture
Source: Attention Is All You Need [5]

2.6.1 Inputs embeddings

As mentioned earlier in this document, computers are not able to process words straight away, they need to be expressed as vectors or matrices in order to be interpreted. The main idea is to map each word to a point in space where words with similar meanings are spatially close to each other. This is often referred to as "Embedding space".

2.6.2 Positional encoding

The vectorisation of words in itself is not enough; the same word in different contexts may have completely different meanings. That is where positional encoders come in.

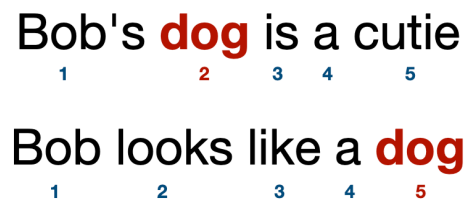

Figure 16: The same word could have different meanings depending on the context

Since transformers contain no recurrence and no convolution, for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, we add "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks [5]. The positional encodings have the same dimension as the embeddings so that the two can be summed.

There are many choices of positional encodings, In the original paper [5], they use sine and cosine functions of different frequencies. In the below expression, we can see that even positions correspond to the sine function and odd positions correspond to the cosine function.

$$PE_{(k,2i)} = sin(k/n^{\frac{2i}{d}})$$

$$PE_{(k,2i+1)} = cos(k/n^{\frac{2i}{d}})$$

Being

k the position of a word in the sequence

i used for mapping to column indices $0 \leq i \leq \frac{d}{2}$

n a User-defined scalar. Originally set to 10000.

d the dimensionality of input and output

$PE_{(pos,2i)}$ the positional encoding for even column indices positions

$PE_{(pos,2i+1)}$ the positional encoding for odd column indices positions

The following figure shows an example of Positional Encoding for the sentence "I am a student" with $n = 100$ and $d = 4$.

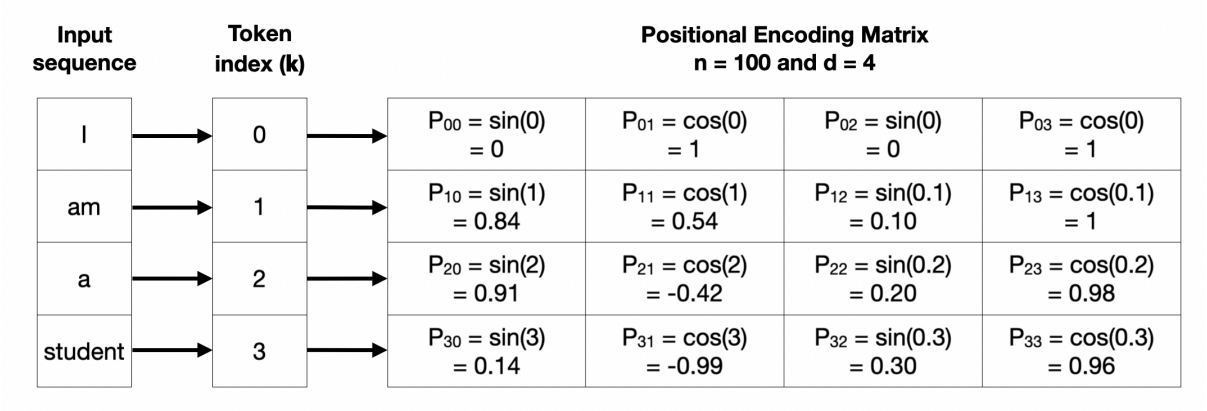

Figure 17: Example of Positional Encoding Matrix Calculation

2.6.3 Self-attention mechanism

Attention implies which part of the sentence to focus on [5]. An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query (Q), keys (K), values (V) and output are all vectors. These vectors are created by multiplying the embedding by three weights matrices (W_Q, W_K and W_V respectively) that are adjusted during the training process. Notice that these new vectors could be smaller (architecture choice) in dimension than the embedding vector.

$$Q = X \times W_Q$$

$$K = X \times W_K$$

$$V = X \times W_V$$

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

Being

Q the query vector

K the keys vector

V the values vector

d_k the dimensionality of K

So far, the attention mechanism has provided one answer at once for each query. The transformer model introduces Multi-Head attention [5], which extends the previous mechanism to produce an answer that combines multiple key-query comparations. Multi-Head attention consists of performing several parallel attention operations and combining outputs to obtain the final context vector. Each attention operation (or "head") is performed by applying a linear projection to the query, keys and values, then calculating the attention between them and subsequently projecting the result to a common space.

$$MultiHead(Q, K, V) = Concat(head_1, head_2, \ldots, head_h)W^O$$

$$\textbf{where } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

Being

Q the query vector

K the keys vector

V the values vector

h the number of attention layers. Originally set to 8

$head_i$ the i^{th} head

W^O global weigth matrix

W_i^Q the i^{th} query weigth matrix

W_i^K the i^{th} key weigth matrix

W_i^V the i^{th} value weigth matrix

2.6.4 Layer normalisation

It should be noted that each sub-layer (self-attention and FeedForward) of each encoder has a residual connection around it, and is followed by a layer normalisation step [5].

It consists of normalising the outputs of the neural network layers so each neurone follows a normal distribution. To normalise a layer, we calculate the standard deviation and the mean of each neurone's activations in that layer, then subtract the mean and divide it by the standard deviation.

The normalisation of a layer can be calculated as follows:

$$LayerNorm(a_i) = \gamma \frac{a_i - \mu_i}{\sigma_i} + \beta$$

Being

a_i the i^{th} layer

μ_i the mean vector of the i^{th} layer

σ_i the standard deviation vector of the i^{th} layer

Note that γ and β are parameters learned during training, and are used to enable the layer to generate normal distributions different from the standard normal distribution.

2.6.5 Pre-trained models

Transformers are replacing RNN-based models and becoming the primary choice for NLP problems. Due to its facility to perform parallelisation, it allows training on bigger datasets. This resulted in the development of pre-trained models such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer), which are trained on large linguistic datasets and can be fine-tuned for specific tasks.

2.6.5.1 BERT

Means Bidirectional Encoder Representations from Transformers [11], BERT's main technical innovation is to apply the bidirectional training of Transformer, this contrasts with previous efforts, which examined a sequence of text from left to right or combined left-to-right and right-to-left training.

To make BERT handle various downstream tasks, this input representation is able to unambiguously represent both a single sentence and a pair of sentences (e.g., Question, Answering) in one token sequence. In the original paper, a “sentence” can be an arbitrary span of contiguous text, rather than an actual linguistic sentence. A “sequence” refers to the input token sequence to BERT, which may be a single sentence or two sentences packed together. The first token of every sequence is always a special classification token ([CLS]). The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks. Sentence pairs are packed together into a single sequence.

The input embedding is denoted as E, the final hidden vector of the special [CLS] token $C \in R^H$, and the final hidden vector for the i^{th} input token $T_i \in R^H$. For a given token, its input representation is constructed by adding the corresponding token, segment, and position embeddings.

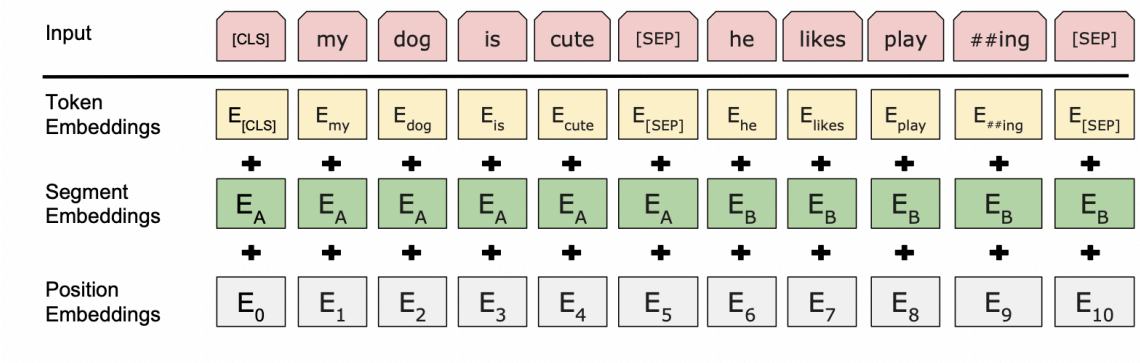

Figure 18: BERT input representation
Source: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [11]

This model consists of two steps: pre-training and fine-tuning. During pre-training, the model is trained with unlabelled data on different tasks. During the tuning step, the model is initialised with the pre-trained parameters, and all of these are refined using labelled data for subsequent tasks. Furthermore, each of these tasks has different tuning models.

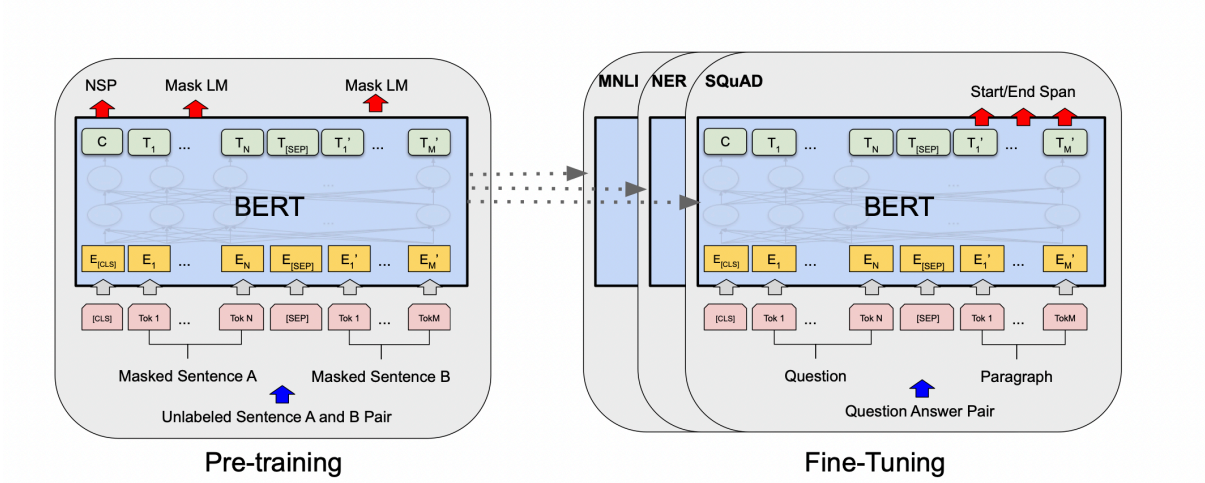

Figure 19: BERT overview

Source: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [11]

The architecture of the BERT model consists of a multi-layer bi-directional transformer encoder based on the original implementation but removing the decoder from the model and just keeping the encoder part.

BERT has two model sizes. Firstly, there is BERT Base, with 12 blocks, 768 hidden layers and 12 heads, so the total number of parameters is 110 million. If this is not enough, there is an extended version of this model known as BERT Large, with 24 blocks, 1024 hidden layers and 16 heads, bringing to 340 million the total number of parameters.

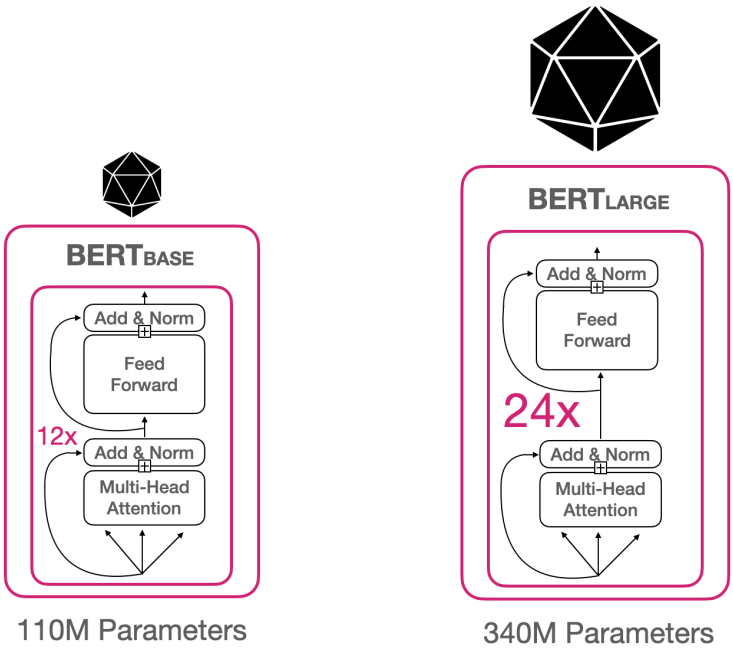

Figure 20: BERT sizes
Source: Hugging Face Inc[3]

2.6.5.2 DistilBERT

DistilBERT is a small, fast, cheap and light Transformer model trained by distilling the BERT base [22]. It has 40% fewer parameters than BERT Base and runs 60% faster while retaining 97% of BERT's performances as measured on the GLUE language understanding benchmark.

Model	**Score**	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
ELMo	68.7	44.1	68.6	76.6	71.1	86.2	53.4	91.5	70.4	56.3
BERT-base	79.5	56.3	86.7	88.6	91.8	89.6	69.3	92.7	89.0	53.5
DistilBERT	77.0	51.3	82.2	87.5	89.2	88.5	59.9	91.3	86.9	56.3

Figure 21: DistilBERT performance comparison
Source: DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter [22]

[3] https://huggingface.co/blog/bert-101

2.6.5.3 RoBERTa

Based on the BERT Base model, which means: “Robustly Optimised BERT Pre-training Approach” [24], It builds on BERT and modifies key hyper-parameters, removes the next-sentence pre-training objective and trains with much larger mini-batches and learning rates. RoBERTa uses a byte-level BPE (Byte Pair Encoding) as a tokeniser (same as GPT-2) and a different pre-training scheme.

2.6.5.4 GPT-2

This is a transformer model trained on a very large corpus of English data using a self-supervised approach [16]. This means that it has been trained on the raw texts only, without being labelled in any way by humans and unlike BERT, this model is unidirectional.

More precisely, inputs are sequences of the continuous text of a certain length and the targets are the same sequence, shifting one token (word or piece of word) to the right. The model uses internally a masking mechanism to make sure the predictions for a token only use the previous inputs but not the future ones. This way, the model learns an inner representation of the English language that can then be used to extract features useful for downstream tasks.

2.7 Contributions

Two possible modifications are then proposed with the intention of increasing the accuracy of the task of text classification in the previously described models. Both approaches rely on assisting the models by adding a significant number of additional layers to the model output.

2.7.1 Fully connected approach

It consists of adding seven linear layers at the end of a pre-trained model instead of one in order to perform a better classification by not reducing so drastically the output of it, as could be the case of

BERT which returns 768 outputs [11] in one shot to only 4 or 6, matching the number of classes ($|C|$) to predict.

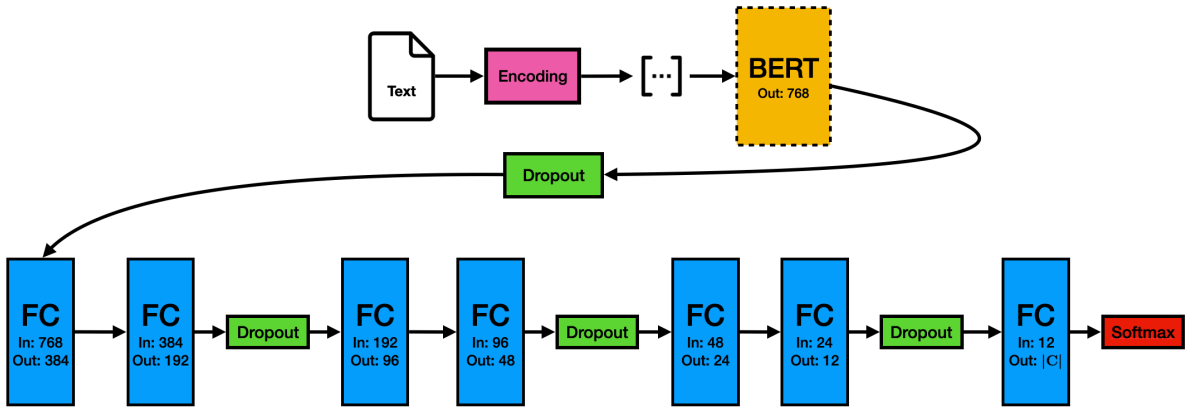

Figure 22: Proposed fully connected network appendix with pre-trained BERT model

2.7.2 Convolutional approach

This modification attempts to translate an NLP problem into a Computer Vision one. Therefore, the idea is to transform the output of the BERT model into an image-like output and apply convolutional layers as if it were an image recogniser.

Given the computational resources and the available pre-trained models, it has been opted to transform the output of these models to black and white 16x16 pixel images (1x16x16) obtaining 256 values per sample.

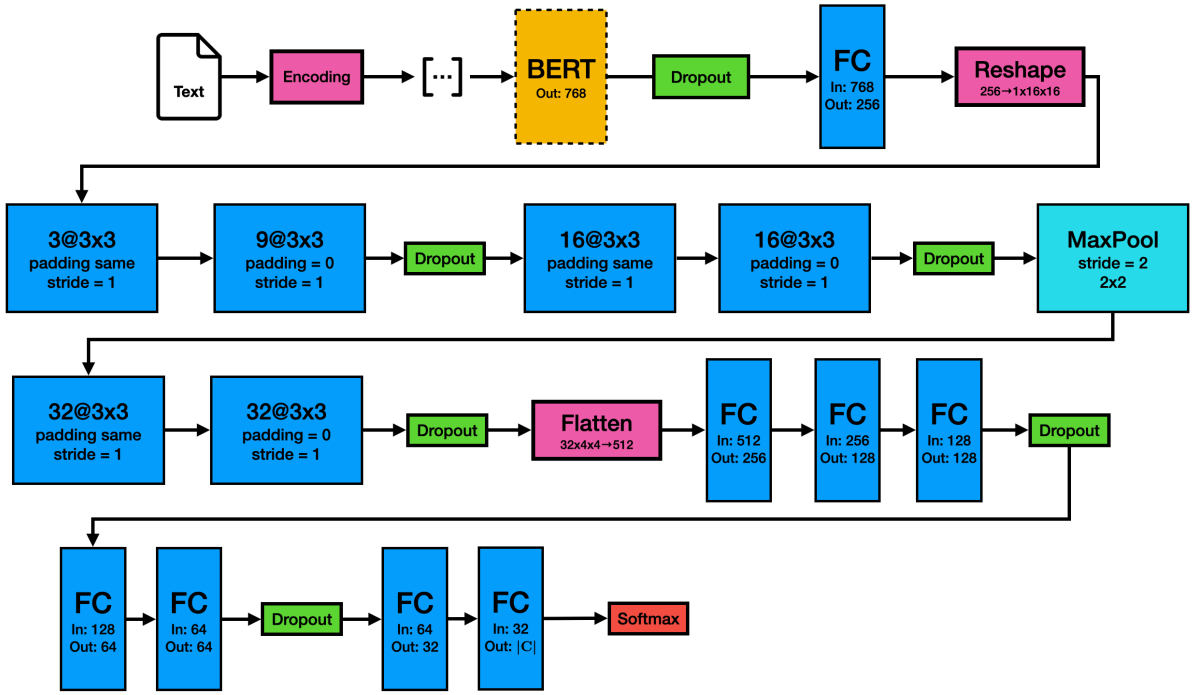

Figure 23: Proposed convolutional network appendix with pre-trained BERT model

Chapter III - Experimental framework

This section describes the experimental framework that is to be used to carry out the study and the necessary preprocessing of the available data. First of all, the format of the data and its origin, any necessary preprocessing, encoding, evaluation metrics and parameters will be described. With all these preliminary steps, it is intended to solve the problem of text classification in the best possible way.

3.1 Dataset

To be able to train the model, it will be necessary that the employed dataset complies with the following structure:

Processed Dataset Structure		
Class	Heading	Article body
1	OTT is the best thing that has happened for young talented people: Bobby	Talking about OTT platforms, Bobby Deol said that it is the best thing that has happened for young talented…
0	Tesla opens its largest Supercharger station in Shanghai	Tesla opens its largest Supercharger station in Shanghai China, with 72 charging stalls…
4	Day 2 play abandoned due to rain, India trail Australia by 307 runs	The second day's play of the Brisbane Test was prematurely abandoned due to rain…
...	...	...

Table 1: Example format for input data files

Regardless of the chosen dataset, it needs to be split into three subsets: training, test and development. If this is not the case, it would have to be rearranged. This is necessary because we have to continuously check the model does not memorise the training data. We do this by evaluating the model with the development set at each epoch. But at the same time to avoid the model fitting too closely to the development set, the last validation will be against the test set and this will be the metric used for benchmarking.

3.1.1 Inshort-news corpus

Indeed, this corpus[4] has 12120 news samples collected from the Inshorts web app, a platform which aggregates news in a limited amount of words. This makes it ideal for classifying news. From the total news provided by this corpus, 8726 have been used for training, 970 for development and 2424 for model evaluation.

However, there are currently no previously existing experiments for this task to compare with. Instead, it has been decided to do all experiments using a well-known dataset and then extrapolate the obtained results to the original dataset.

3.1.2 AG News corpus

The alternative task under consideration is "AG News Classification" also available on the Kaggle platform[5]. It contains 4 classes (fewer than the previous set) distributed over 127,600 samples. This is 10.5 times more samples than the Inshort set.

AG is a compilation of well over 1 million news articles. In more than 1 year of activity, ComeToMyHead has collected news articles from more than 2000 news sources. It is a search engine for academic news that has been running since July 2004.

---

[4] https://www.kaggle.com/datasets/kishanyadav/inshort-news

[5] https://www.kaggle.com/datasets/amananandrai/ag-news-classification-dataset

The dataset is provided by the academic community for research purposes in data mining, information retrieval, data compression, data streaming and any other non-commercial activity. The AG news classification dataset is built by choosing the 4 largest classes from the original larger corpus.

This dataset is really large compared to our available hardware. This makes model training and evaluation much more time-consuming. On the other hand, having more data means learning could potentially be better.

Available tags	
Inshort-news	**AG News corpus**
automobile, entertainment, politics, science, sports, technology, world.	business, sci/tech, sports, world.

Table 2: Available classes

3.2 Pre-processing

To make the experimentation simpler, both datasets have been pre-processed. By doing so, we ensure that the structure is the same and we can swap them quickly.

Each dataset has been rearranged into three subsets: training, development and test. This will be done once to ensure all future experiments use the same data and thus compare them fairly. Additionally, it can be guaranteed that in all subsets there is the same proportion of classes as in the full set.

3.2.1 Inshort-news corpus

Initially, this set is provided as seven separate files, one per class. As a result, this turns out to be inconvenient, as we need the samples to be merged and mixed with others of different classes to be able to use it.

Later on, it will also be necessary to make a partition of the dataset to fit the previously established division.

3.2.2 AG News corpus

This set comes initially split into 2 subsets: train and test. Again, it does not fit the requirements set out above and needs to be mixed and separated back in the same way as with the Inshort-news corpus. This results in a division of 108 000 samples for training, 12 000 samples for development and 7 600 samples for test.

Class distribution			
Inshort-news corpus			
All	Train	Development	Test
automobile 10.67% entertainment 16.8% politics 13.17% science 11.86% sports 15.68% technology 14.78% world 17.05%	automobile 10.69% entertainment 16.87% politics 13.35% science 11.78% sports 15.6% technology 14.76% world 16.95%	automobile 11.65% entertainment 16.8% politics 11.75% science 11.65% sports 14.74% technology 14.23% world 19.18%	automobile XX% entertainment 16.54% politics 13.08% science 12.21% sports 16.34% technology 15.06% world 16.58%
AG News corpus			
All	Train	Development	Test
business 25.0% sci/tech 25.0% sports 25.0% world 25.0%	business 24.97% sci/tech 24.97% sports 25.06% world 25.0%	business 25.25% sci/tech 25.27% sports 24.50% world 24.98%	business 25.0% sci/tech 25.0% sports 25.0% world 25.0%

Table 3: Distribution of the sample classes after partitioning

To guarantee a correct model training, as far as possible the same proportion of classes in each split should be used to avoid biases.

3.3 Tokenisation

To be able to feed the model with strings, it is necessary to tokenise them first [1]. This process consists of chunking raw text into small parts. Tokenisation breaks the raw text into words and phrases called tokens [13]. These tokens help understand the context. It allows interpreting the meaning of the text by analysing words in sequence.

There are different methods (e.g Moses[6]) and libraries available for tokenisation such as NLTK[7] or Gensim[8]. However, because we are using pre-trained models provided by Hugging Face, Inc. Each model has its own ad hoc tokeniser specifically designed for it, so it is more advisable to use them rather than other alternatives.

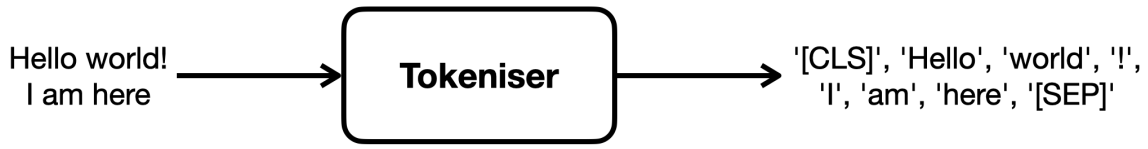

Figure 24: Tokeniser expected behaviour

Setting an appropriate value for the token length parameter is crucial for the model's performance. Although models such as BERT and its derivatives have shown state-of-the-art results in most areas of NLP, the Transformer model has a major handicap: it is difficult to apply to very long texts. This difficulty is due to the self-attention operation, which has an exponential complexity of $O(n^2)$ relative to the input length. This can be problematic, as not everyone has the resources to efficiently handle the size and complexity of the data. Some examples are novels, legal documents, medical records, etc.

[6] http://www2.statmt.org/moses/

[7] https://www.nltk.org

[8] https://pypi.org/project/gensim/

Several solutions have been proposed to mitigate the problem with long documents. The simplest one is truncation, where only the first N tokens of the documents are taken as input, but this wastes a lot of potentially valuable information from the original source. For the news classification task, this approach can be perfectly suitable if we do a proper token length analysis.

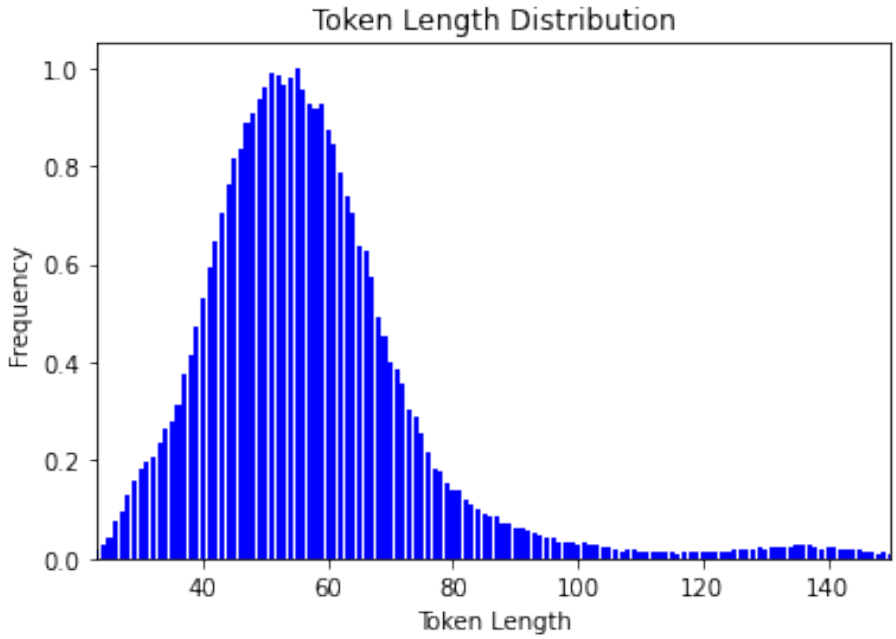

Figure 25: Token length distribution.

As shown in the figure above, a token length of 100 would be enough to represent the news from any of the datasets.

3.4 Encoding

Linked to the previous section, it is equally imperative to perform the encoding of the tokenised words beforehand, in other words, to give them a vector shape [1]. It can be achieved through a variety of available tools and methods.

Such as:

- **Frequency-based Embedding**: Count Vectors, TF-IDF, Co-Occurrence Matrix [19].
- **Prediction-based Embedding (Word2vec)**: CBOW, Skip-Gram [19].
- **GloVe**: Global Vectors for Word Representation [12].

Computers only understand numbers, so a good tokeniser should establish semantic relationships between words through the use of vector operations.

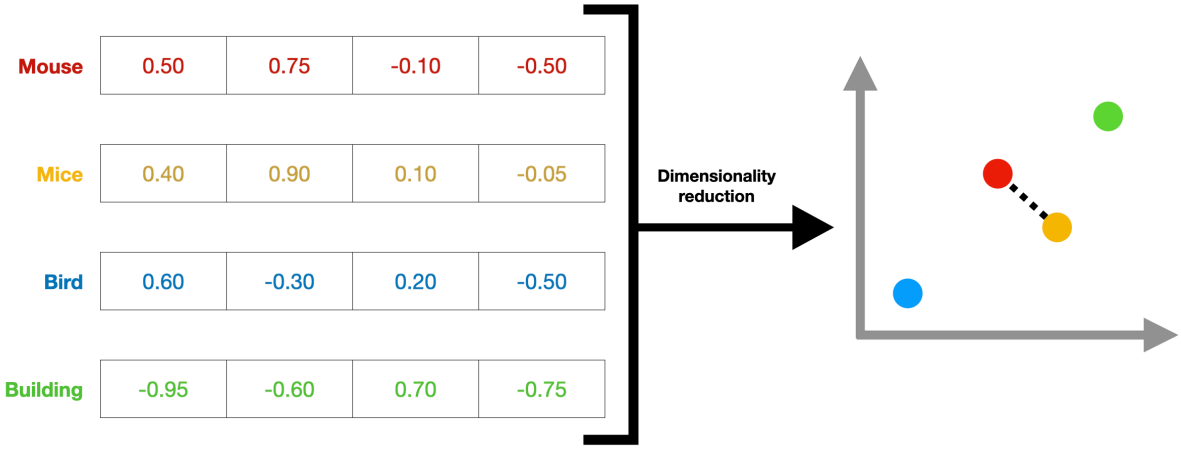

Figure 26: How a vectoriser work

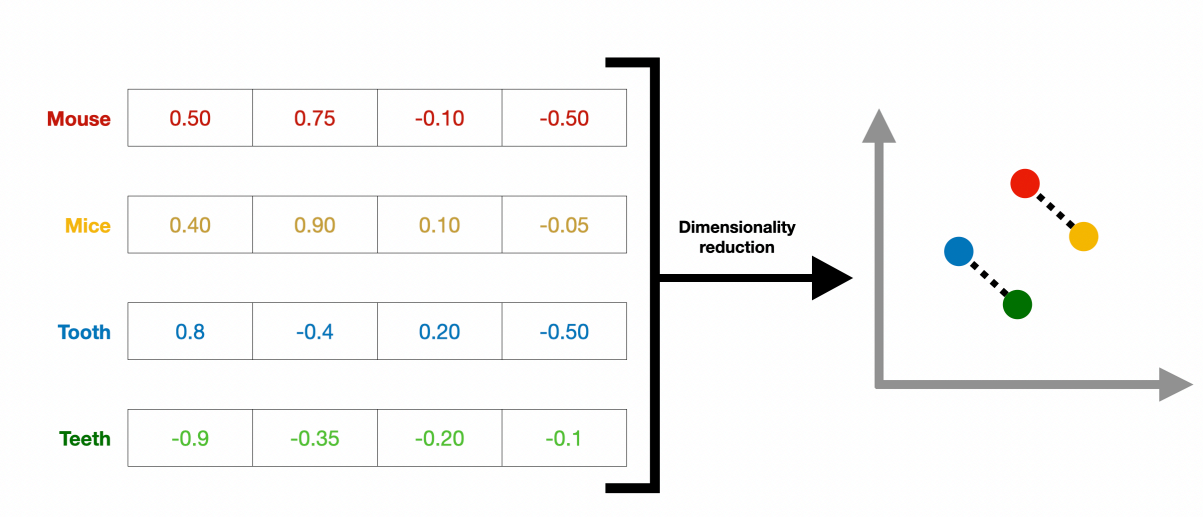

Figure 27: How a vectoriser makes semantical relationships

Since we are using pre-trained models, the Hugging Face, Inc 'transformers' library provides a specific vectoriser within each tokeniser. Therefore, for the same reason as in the previous section, this one will be used instead of any other third-party solution.

3.5 Metrics

For the evaluation of NLP models, a variety of metrics are used to measure their quality. In the case of automatic text classification, the following metrics have been used:

- Accuracy
- Loss
- Precision
- Recall
- F1 Score

3.5.1 Accuracy

Measures the number of well-classified samples relative to the total amount of them.

$$\text{accuracy} = \frac{TP}{Total}$$

Being

TP true positive values

$Total$ all available samples

3.5.2 Loss

Cross-entropy loss measures the performance of a classification model whose output is a probability value in the (0, 1) range. In multi-class classification, a separate loss is calculated for each class label per observation and the result is then added up.

$$L_{o,c} = -\sum_{c=1}^{M} y_{o,c} \log p_{o,c}$$

Being

M number of classes

$y_{o,c}$ binary indicator (0 or 1) if class label c is the correct for observation o

$p_{o,c}$ predicted probability observation o is of class c

3.5.3 Precision

This metric measures how accurate a model is by establishing a relationship between the number of true positives and all identified as positives.

$$\text{precision} = \frac{TP}{TP + FP}$$

Being TP true positive values and FP false positive values.

3.5.4 Recall

Measures the model's ability to predict true positive classes. It is the ratio between the predicted true positives and what was labelled. The recall metric reveals how many predicted classes are labelled well.

$$\text{recall} = \frac{TP}{TP + FN}$$

Being

TP true positive values

FN false negative values

3.5.5 F1 Score

This is a Precision and Recall function. It is necessary when a balance between precision and recall is desired.

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Being

TP true positive values

FP false positive values

FN false negative values

3.6 Parameters

Each model has a set of hyper-parameters that should be adjusted to achieve better results. The corresponding parameters are indicated in the table below.

Parameters			
	Name	**Type**	**Description**
Common	Learning rate	Float	Initial learning rate before scheduling
	Epochs	Integer	Number of training set iterations
	Batch size	Integer	Number of training samples per batch
	Optimiser	Object	Model optimiser (Adam, SGD, etc)
	Maximum token length	Integer	First N tokens to be evaluated per sample
Transformer based	Frozen	Boolean	Freeze pre-trained model or not
LSTM based	Maximum word vocab.	Integer	Maximum vocabulary available.
	Hidden dimension	Integer	Hidden size of the LSTM layer
	Number of layers	Integer	Number or LSTM layers.

Table 4: Adjustable parameters for each model type

3.7 Environment

For the experiments, two NVIDIA GTX 1080 GPUs with CUDA version 10.1 are hosted on a dedicated PRHLT server powered by Ubuntu 20.04.

In order to speed up the experiments running others in parallel, an additional computer, a Dell XPS 15 9570 equipped with an Intel core i7-8750H processor, 16GB RAM and Nvidia 1050 4GB NVRAM Max-Q GPU, is used. In any case, all machines have the same dependencies installed in a CONDA environment.

The essential python modules for running the experiments are PyTorch v1.11, Torchaudio v0.11, Torchvision v0.1, Transformers v4.18, Hugging face Hub v0.5.1, Numpy v1.19 and Pandas v1.4.2.

Chapter IV - Experiments

In this study, various transformer-based neural models have been tested, however, it is also desirable to put these results in perspective with the previous state-of-the-art models, i.e. the recurrent models with attention mechanisms or LSTMs.

To ensure fair model comparisons, the following restrictions are established:

- All models have the same random seed.
- All models share the same training, test and set partitions.
- Each model must run for no more than 100 epochs. In the case of transformer-based models, an additional 100 epochs are allowed for fine-tuning.
- Every model has $2 \cdot 10^{-5}$ as the initial learning rate. For transformer-based models, the initial value of the learning rate is reduced to $2 \cdot 10^{-8}$ in the fine-tuning process.

The results in the following tables refer to the best version of each model evaluated over the test set. The best version of a model is defined as that which provides the highest accuracy value during the model training against the development set.

Every model implementation in this study follows the same scheme: the text is processed directly to the base model (LSTM, BERT, etc.) and then the last hidden state generated by that model is extracted and passed through a fully connected layer for classification. With the exception of the last models, which are proposed contributions that have a significantly higher number of layers than the previous ones.

4.1 LSTM

Long Short-term memory (LSTM) is a kind of artificial neural network. Unlike standard networks, the LSTM has feedback connections. While recurrent neural networks (RNN) can process individual data points and entire sequences of data, the LSTM architecture is intended to provide RNNs with a short-term memory that can last for thousands of time steps, so it is a "long-term memory" [17]. A common LSTM unit consists of a cell, an input gate, an output gate and a forgetting gate. The cell remembers values during arbitrary time intervals and the three gates regulate the flow of information in and out of the cell. The following tables of results have been obtained for the LSTM model.

		Predicted values				
	12000 samples	**World**	**Sports**	**Business**	**Sci/Tech**	*Macro Recall*
Actual values	**World**	2541	134	165	158	84.76%
	Sports	66	2787	16	70	94.83%
	Business	153	18	2494	365	82.31%
	Sci/Tech	173	56	273	2531	83.45%
	Macro Precision	86.63%	93.06%	84.60%	81.02%	

Table 5: Confusion matrix of LSTM on development data

As can be seen, this model is able to differentiate quite certainly considering its execution time of just 3 minutes per epoch. It is also remarkable its ability to differentiate sports news from the other categories. There is a difference of approximately 10% on average over the others.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
86.28%	86.33%	86.34%	86.33%

Table 6: Average LSTM macro-metrics on development data

This model sets the baseline for the rest of the models that will be developed later in this document. In the table above it can be seen that we start from approximately 86.30% in all macro metrics. In this case, they are so similar and it is a good indicator. If we have to choose between similar models that differ in precision and/or recall, the best trade-off will be the one with the best balance between them, As known as F1-Score.

The table below shows the rate of True Positives, True Negatives, False Positives and False Negatives for each class in the dataset.

12000 samples	True Positive (TP)	True Negative (TN)	False Negative (FN)	False Positive (FP)
World	2541	8610	457	392
Sports	2787	8853	152	208
Business	2494	8516	536	454
Sci/Tech	2531	8374	502	593

Table 7: Detailed confusion matrix for LSTM with development data for every single class.

In the following figure, we can see how the accuracy of the model evolves in the training and development set. It can be seen that the evolution of the development set mirrors the training set relatively well. However, we can appreciate that there is a certain decoupling between them, especially in the intervals (10, 40) and at the end of the execution. With slight overfitting.

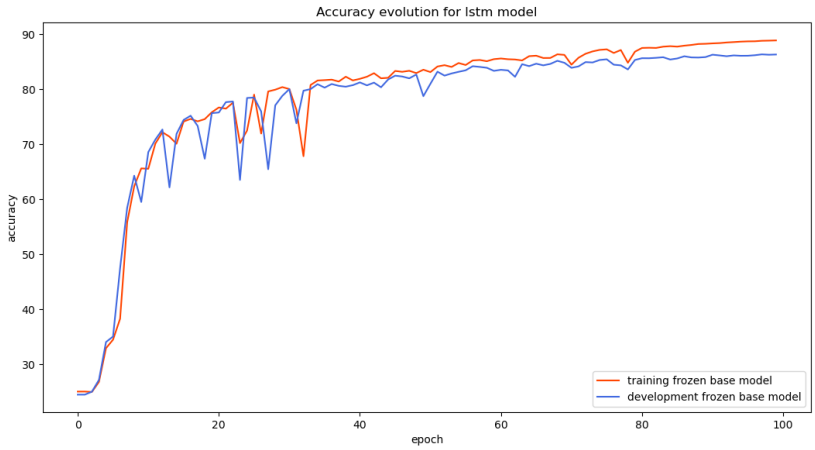

Figure 28: Dev accuracy evolution for LSTM model

4.2 Bidirectional LSTM

LSTM is a Gated Recurrent Neural Network [17], and bidirectional LSTM is just an extension of it. The main feature is that those networks can store information that can be used for future cell processing. In bidirectional LSTM, instead of just training one model, we use two [14]. The first model learns the sequence from the provided input, the second model learns this sequence backwards. The following tables of results have been obtained for the Bi-directional LSTM model.

		Predicted values				
	12000 samples	**World**	**Sports**	**Business**	**Sci/Tech**	*Macro Recall*
Actual values	**World**	2592	126	165	115	86.46%
	Sports	38	2845	31	25	96.80%
	Business	99	29	2619	283	86.44%
	Sci/Tech	92	34	233	2674	88.16%
	Macro Precision	91.88%	93.77%	85.93%	86.34%	

Table 8: Confusion matrix of Bidirectional LSTM on development data

According to the table above, the Bidirectional LSTM network improves on its predecessor in all aspects. It is also noticeable that it still maintains its capacity to significantly distinguish sports news from the rest.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
89.42%	89.48%	89.46%	89.47%

Table 9: Average Bidirectional LSTM macro-metrics on development data

As can be seen, the use of bidirectional recurrent neural networks with attention mechanisms in this problem implies (as expected) a remarkable improvement in the classification. Being able to reach the 90% accuracy threshold in 6.5 minutes per epoch. Despite taking twice the time of the standard LSTM network, it still takes a fairly reasonable amount of computing time per forward considering the actual performance. Particularly if low-resource hardware is used in the deployment.

The table below shows the rate of True Positives, True Negatives, False Positives and False Negatives for each class in the dataset.

12000 samples	True Positive (TP)	True Negative (TN)	False Negative (FN)	False Positive (FP)
World	2592	8773	406	229
Sports	2845	8872	94	189
Business	2619	8541	411	429
Sci/Tech	2674	8544	359	423

Table 10: Detailed confusion matrix for Bidirectional LSTM with development data for every single class.

In the figure below, we can see how the accuracy of the model evolves in the training and development sets. It can be seen that the evolution of the development data almost perfectly mirrors the results obtained in the test set during the first 20 iterations. From then on, there is a slight divergence that widens over time, which

indicates that the overfitting is much more intense than in the previous model.

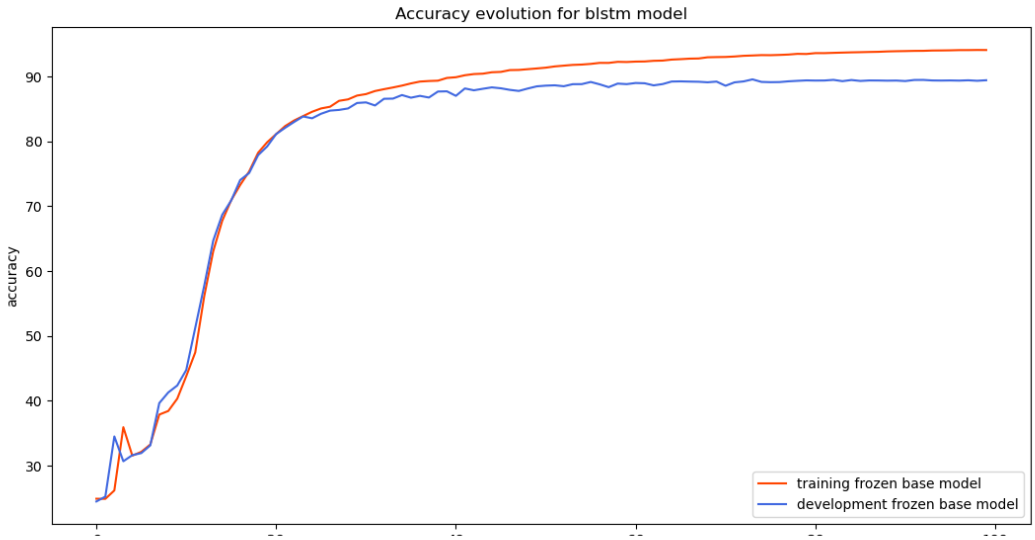

Figure 29: Dev accuracy evolution for Bidirectional LSTM model

4.3 BERT

Means Bidirectional Encoder Representations from Transformers, BERT's main technical innovation is to apply the bidirectional training of Transformer [11], this contrasts with previous efforts, which examined a sequence of text from left to right or combined left-to-right and right-to-left training. The presented data refers to the fine-tuned version of the pre-trained BERT model. The following tables of results have been obtained for the BERT model.

		Predicted values				
	12000 samples	**World**	**Sports**	**Business**	**Sci/Tech**	*Macro Recall*
Actual values	**World**	2671	109	125	93	89.09%
	Sports	10	2914	11	4	99.15%
	Business	88	15	2646	281	87.33%
	Sci/Tech	64	17	169	2783	91.76%
	Macro Precision	94.28%	95.38%	89.66%	88.04%	

Table 11: Confusion matrix of BERT on development data

Moving to transformer-based models such as BERT, we see a considerable improvement in the results. While the business and sci/tech categories are still confused, it is less common. This is understandable in part because science and technology are in many cases the main drivers of the world's economic development nowadays.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
91.78%	91.84%	91.83%	91.84%

Table 12: Average BERT macro-metrics on development data

If we look at the macro-metrics in the table above, we can see that BERT is the first model, of those seen so far, to surpass 91% accuracy. However, this is not cost-free due to the enormous cost

of forwarding to it without a graphics card. In this case, with an Nvidia 1080 GPU, each epoch took around 25 minutes, and in order to achieve these results, an extra 100 iterations of fine-tuning were required. So we should seriously consider whether the improvement in accuracy is worth the increased use of resources.

The table below shows the rate of True Positives, True Negatives, False Positives and False Negatives for each class in the dataset.

12000 samples	True Positive (TP)	True Negative (TN)	False Negative (FN)	False Positive (FP)
World	2671	8840	327	162
Sports	2914	8920	25	141
Business	2646	8665	384	305
Sci/Tech	2783	8589	250	378

Table 13: Detailed confusion matrix for BERT with development data for every single class.

In the figure below, we can see how the accuracy of the model evolves in the training and development sets. It can be observed that the evolution of the development data perfectly replicates the evolution of the training set. This behaviour is mirrored in the first training stage and the fine-tuning phase. So we can assume that this model does not suffer from overfitting. But what we can say with certainty is that this model is the most stable of those seen so far in this document in terms of its metrics during training.

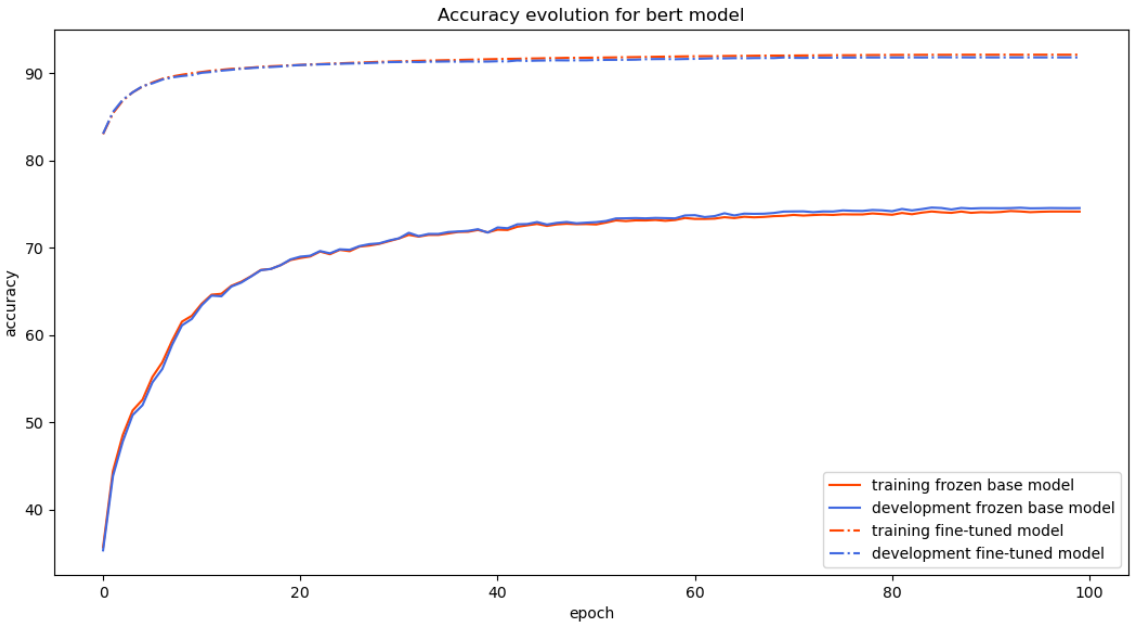

Figure 30: Dev accuracy evolution for BERT model

4.4 BERT uncased

This is a modification of the original BERT model, with the special characteristic that it does not differentiate between upper and lower case letters. The following tables of results have been obtained for the BERT uncased model.

		Predicted values				
	12000 samples	**World**	**Sports**	**Business**	**Sci/Tech**	*Macro Recall*
Actual values	**World**	2586	131	156	125	86.26%
	Sports	79	2807	30	23	95.51%
	Business	126	36	2557	311	84.39%
	Sci/Tech	111	41	234	2647	87.27%
	Macro Precision	89.11%	93.10%	85.89%	85.22%	

Table 14: Confusion matrix of BERT uncased on development data

Unfortunately, this variant of BERT worsens the results obtained by the original model in every aspect, being even poorer than a bidirectional LSTM network. Therefore, in terms of both computational cost and results obtained, this model would be completely out of the scope of this task in any case. This may be due to using acronyms and abbreviations that are usually capitalised and so typical of news. It would not be surprising if this could confuse the model enough to make its results so much worse.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
88.31%	88.33%	88.36%	88.34%

Table 15: Average BERT uncased macro-metrics on development data

The table below shows the rate of True Positives, True Negatives, False Positives and False Negatives for each class in the dataset.

12000 samples	True Positive (TP)	True Negative (TN)	False Negative (FN)	False Positive (FP)
World	2586	8686	412	316
Sports	2807	8853	132	208
Business	2557	8550	473	420
Sci/Tech	2647	8508	386	459

Table 16: Detailed confusion matrix for BERT uncased with development data for every single class.

The most positive thing about this model which can be seen in the figure below is that like the original BERT it is very stable during training. Coping the results obtained in the development set very well compared to those obtained in the training set. Unfortunately, the BERT model is still better in this aspect as it shows less divergence.

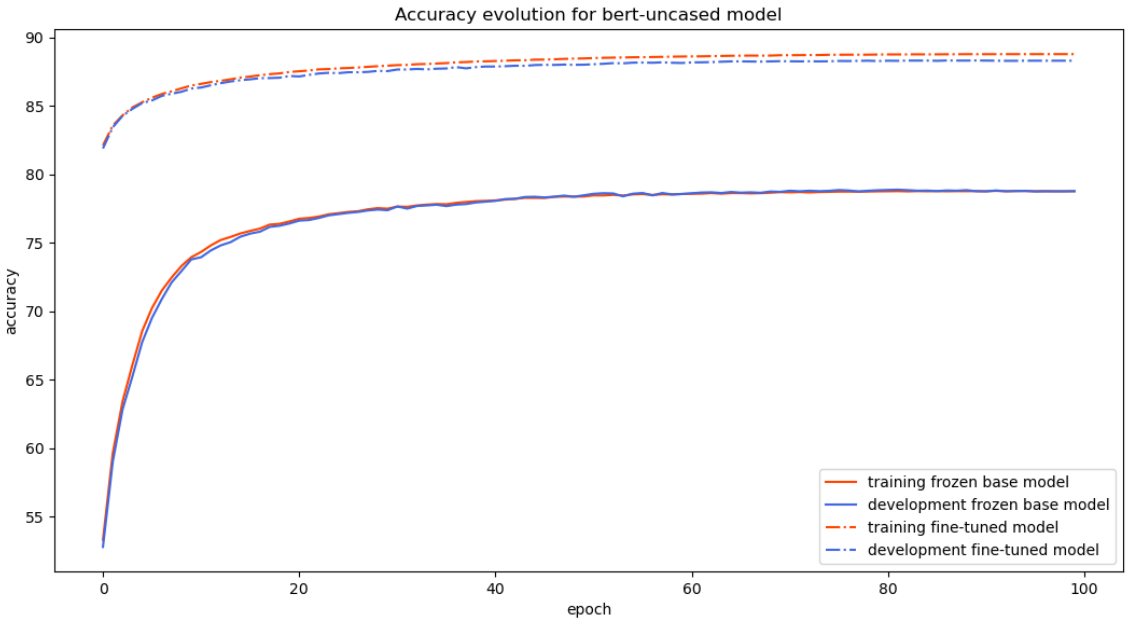

Figure 31: Dev accuracy evolution for BERT uncased model

4.5 DistilBERT

This is another modification of the original BERT model [22], with some changes that make it a smaller, faster, cheaper and lighter version obtained by distilling BERT. The following tables of results have been obtained for the DistilBERT model.

		Predicted values				
	12000 samples	**World**	**Sports**	**Business**	**Sci/Tech**	*Macro Recall*
Actual values	**World**	2563	132	183	120	85.49%
	Sports	68	2813	34	24	95.71%
	Business	116	45	2556	313	84.36%
	Sci/Tech	124	46	250	2613	86.15%
	Macro Precision	89.27%	92.65%	84.55%	85.11%	

Table 17: Confusion matrix of DistilBERT on development data

DistilBERT faces the same problem as BERT uncased, and even worse. While it is true that this model can supposedly retain up to 97% of the capacity of the original BERT model, for this task it is not capable of at least equaling it. This is understandable (compared to BERT uncased) because this model has up to 40% fewer parameters and runs up to 60% faster. In our case, DistilBERT is the model that overcomes the 80% accuracy threshold in the fewest number of iterations, in this case, during the first 20 epochs (before fine-tuning) this model achieves first place.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
87.87%	87.90%	87.93%	87.91%

Table 18: Average DistilBERT macro-metrics on development data

The table below shows the rate of True Positives, True Negatives, False Positives and False Negatives for each class in the dataset.

12000 samples	True Positive (TP)	True Negative (TN)	False Negative (FN)	False Positive (FP)
World	2563	8694	435	308
Sports	2813	8838	126	223
Business	2556	8503	474	467
Sci/Tech	2613	8510	420	457

Table 19: Detailed confusion matrix for DistilBERT with development data for every single class.

In the figure below, we can see how the accuracy of the model evolves in the training and development sets. It can be seen that the evolution of the development data replicates quite well the evolution of the training set, at least in the first phase of training. In the fine-tuning phase, we can observe some de-coupling between the accuracy obtained in the training and development sets. So it suffers even more from overfitting than BERT uncased.

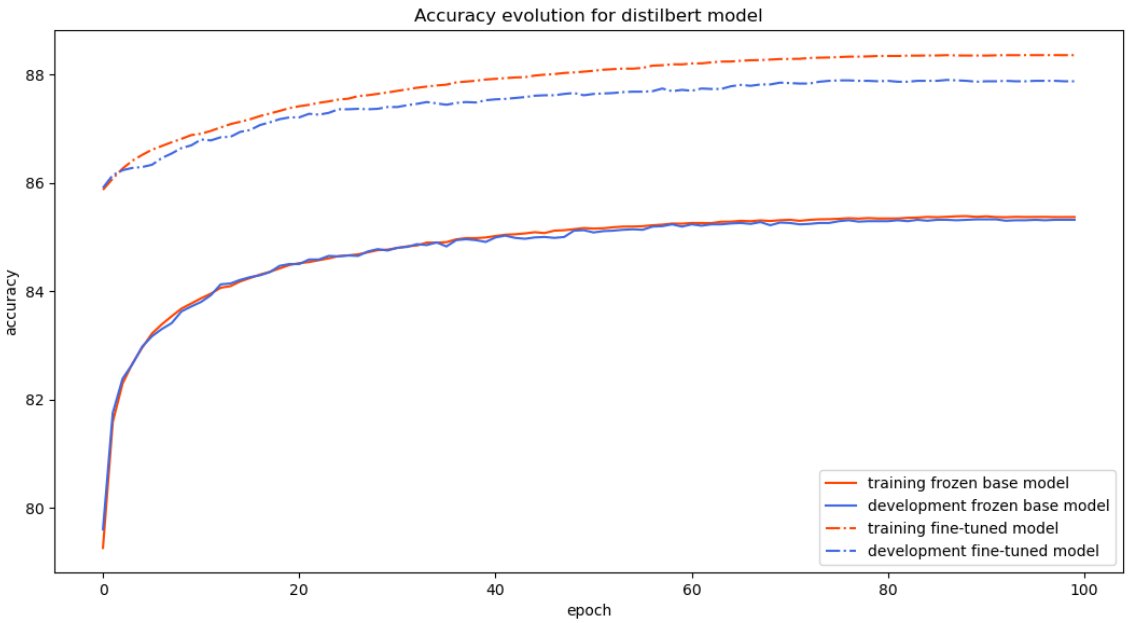

Figure 32: Dev accuracy evolution for DistilBERT model

4.6 RoBERTa

Again, it is a model based on the BERT, which means: “Robustly Optimised BERT Pre-training Approach” [24], It builds on BERT and modifies key hyper-parameters, removes the next-sentence pre-training objective and trains with much larger mini-batches and learning rates. The following tables of results have been obtained for the RoBERTa model.

		Predicted values				
	12000 samples	**World**	**Sports**	**Business**	**Sci/Tech**	*Macro Recall*
Actual values	**World**	2680	111	127	80	89.39%
	Sports	10	2912	9	8	99.08%
	Business	82	9	2679	260	88.42%
	Sci/Tech	76	13	167	2777	91.56%
	Macro Precision	94.10%	95.63%	89.84%	88.86%	

Table 20: Confusion matrix of RoBERTa on development data

As might be expected given the above description, RoBERTa slightly improves on the results obtained by BERT at about the same time cost. In some specific metrics, a very marginal (almost negligible) decrease in some scores can be seen, such as macro recall and macro accuracy for world and sports categories respectively. In general terms, RoBERTa presents the best results in this work.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
92.07%	92.11%	92.11%	92.11%

Table 21: Average RoBERTa macro-metrics on development data

The table below shows the rate of True Positives, True Negatives, False Positives and False Negatives for each class in the dataset.

12000 samples	True Positive (TP)	True Negative (TN)	False Negative (FN)	False Positive (FP)
World	2680	8834	318	168
Sports	2912	8928	27	133
Business	2679	8667	351	303
Sci/Tech	2777	8619	256	348

Table 22: Detailed confusion matrix for RoBERTa with development data for every single class.

Analogous to BERT, the evolution of the accuracy in the development set almost closely mirror those of the training set. Therefore, we can intuitively conclude that this fine-tuning of RoBERTa does not show a high degree of overfitting.

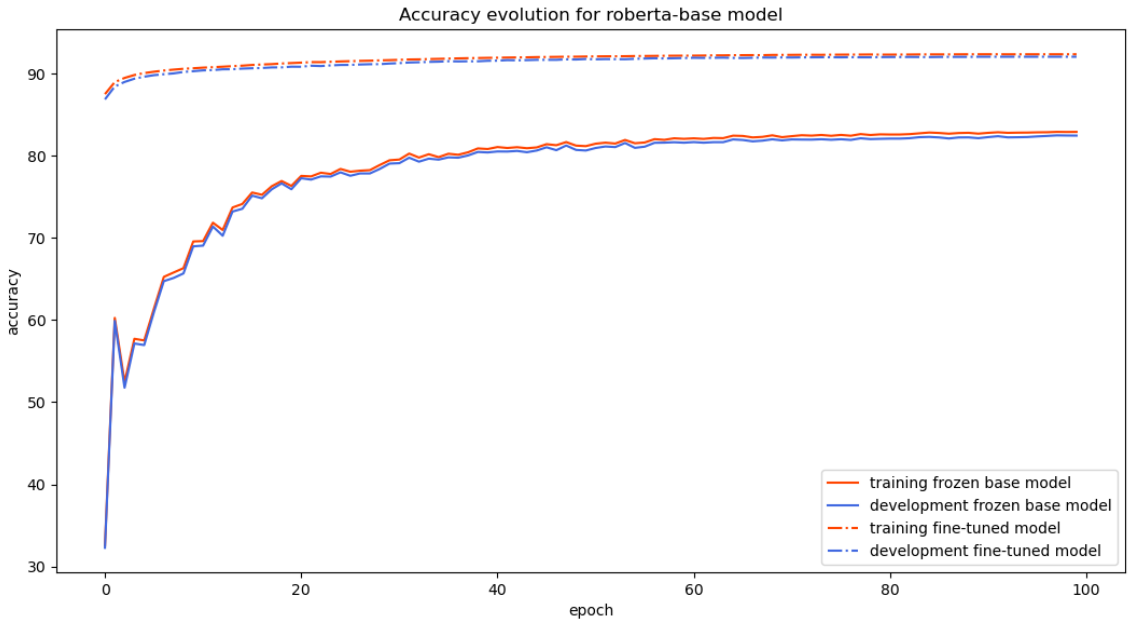

Figure 33: Dev accuracy evolution for RoBERTa model

4.7 XLM-RoBERTa

This is the multilingual version of RoBERTa [4]. It is a large multilingual language model, trained on 2.5TB of filtered CommonCrawl data. Common Crawl is a non-profit organisation that scraps the Internet and publishes its data sets for free. The following tables of results have been obtained for the XLM-RoBERTa model.

		Predicted values				
	12000 samples	**World**	**Sports**	**Business**	**Sci/Tech**	*Macro Recall*
Actual values	**World**	2638	112	157	91	87.99%
	Sports	11	2907	9	12	98.91%
	Business	88	22	2627	293	86.70%
	Sci/Tech	78	18	215	2722	89.75%
	Macro Precision	93.71%	95.03%	87.33%	87.30%	

Table 23: Confusion matrix of XLM-RoBERTa on development data

The results clearly show that having a monolingual dataset significantly detracts from the model's performance. They are worse than the original BERT model but better than those of a bidirectional LSTM network. Probably, in the case of a multilingual dataset, XLM-RoBERTa would have scored better than any other model. So while the current results are disappointing, we cannot discard its capability if we want to deploy this model outside the Anglophone world.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
90.78%	90.84%	90.84%	90.84%

Table 24: Average XLM-RoBERTa macro-metrics on development data

The table below shows the rate of True Positives, True Negatives, False Positives and False Negatives for each class in the dataset.

12000 samples	True Positive (TP)	True Negative (TN)	False Negative (FN)	False Positive (FP)
World	2638	8825	360	177
Sports	2907	8909	32	152
Business	2627	8589	403	381
Sci/Tech	2722	8571	311	396

Table 25: Detailed confusion matrix for XLM-RoBERTa with development data for every single class.

As in its monolingual form, the evolution of accuracy in the development set almost mirrors that of the training set, with no clear evidence of overfitting.

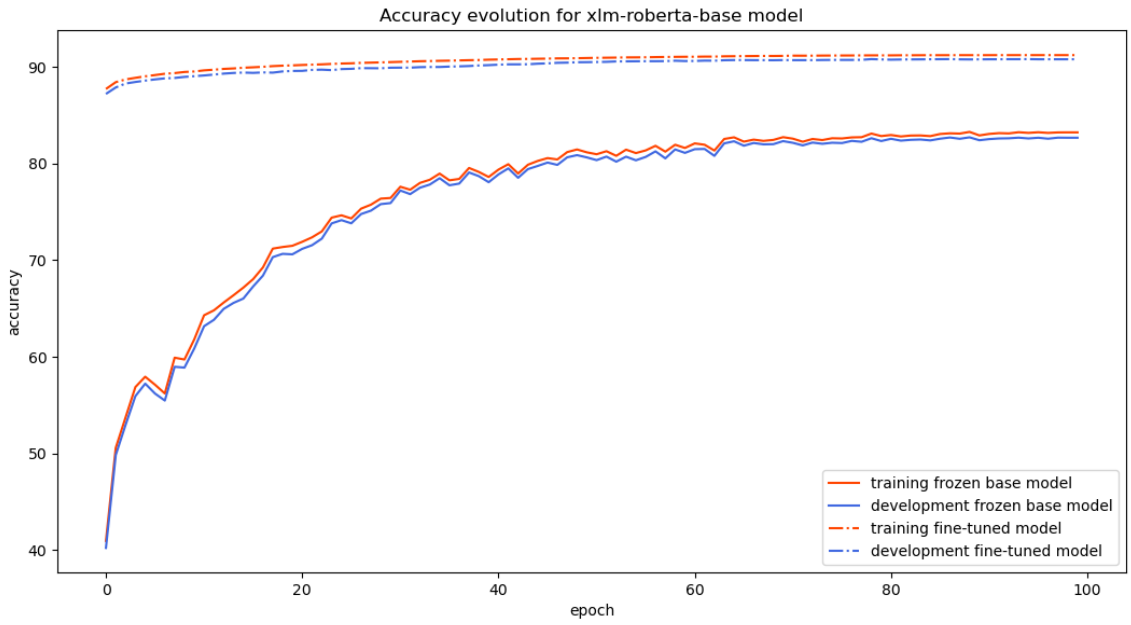

Figure 34: Dev accuracy evolution for XLM-RoBERTa model

4.8 GPT-2

The data presented refer to the fine-tuned version of the pre-trained GPT-2 model. This is a transformer model trained on a very large corpus of English data using a self-supervised approach [16]. This means that it has been trained on the raw texts only, without being labelled in any way by humans. The following tables of results have been obtained for the GPT-2 model.

		Predicted values				
	12000 samples	**World**	**Sports**	**Business**	**Sci/Tech**	*Macro Recall*
Actual values	**World**	1687	665	329	317	56.27%
	Sports	519	1721	342	357	58.56%
	Business	599	766	1099	566	36.27%
	Sci/Tech	433	705	327	1568	51.70%
	Macro Precision	52.10%	44.62%	52.41%	55.84%	

Table 26: Confusion matrix of GPT-2 on development data

There really isn't much to say about this model, for this task the results are absolutely awful, equal to if not worse than tossing dice or flipping a coin. The poor performance of GPT-2 is easily understandable if we consider that this model is decoder-only. That means, in its construction, it omits the fundamental part of a good classifier, the encoder. GPT-2 has a niche in text generation such as translation and question answering, but it is clearly not suitable for classification.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
50.62%	51.24%	50.70%	50.97%

Table 27: Average GPT-2 macro-metrics on development data

The table below shows the rate of True Positives, True Negatives, False Positives and False Negatives for each class in the dataset.

12000 samples	True Positive (TP)	True Negative (TN)	False Negative (FN)	False Positive (FP)
World	1687	7451	1311	1551
Sports	1721	6925	1218	2136
Business	1099	7972	1931	998
Sci/Tech	1568	7727	1465	1240

Table 28: Detailed confusion matrix for GPT-2 with development data for every single class.

As can be seen, it is possible to barely obtain an accuracy close to 50% at best for both the training and the development set.

Figure 35: Dev accuracy evolution for GPT-2 model

4.9 RoBERTa CustomNet

This is the same RoBERTa model described previously with fully connected and dropout layers added after getting the last hidden state from the base model. The following tables of results have been obtained for the RoBERTa CustomNet model.

		Predicted values				
	12000 samples	World	Sports	Business	Sci/Tech	*Macro Recall*
Actual values	**World**	2620	120	148	110	87.39%
	Sports	17	2904	12	6	98.81%
	Business	71	10	2694	255	88.91%
	Sci/Tech	100	22	264	2647	87.27%
	Macro Precision	93.30%	95.03%	86.40%	87.71%	

Table 29: Confusion matrix of RoBERTa CustomNet on development data

This proposal was intended to add several fully connected layers to the original model in the hope of helping the classification task (unsuccessfully) by not reducing the RoBERTa outputs so drastically to the number of labels of the problem. As a result, we have obtained a model not much better than XML-RoBERTa for this monolingual task but with a somewhat higher cost than the original RoBERTa model. Making this approach impracticable for any possible use.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
90.54%	90.61%	90.60%	90.60%

Table 30: Average RoBERTa CustomNet macro-metrics on development data

The table below shows the rate of True Positives, True Negatives, False Positives and False Negatives for each class in the dataset.

12000 samples	True Positive (TP)	True Negative (TN)	False Negative (FN)	False Positive (FP)
World	2620	8814	378	188
Sports	2904	8909	35	152
Business	2694	8546	336	424
Sci/Tech	2647	8596	386	371

Table 31: Detailed confusion matrix for RoBERTa CustomNet with development data for every single class.

As in the original version, the evolution of accuracy in the development set almost mirrors that of the training set, with no clear evidence of overfitting.

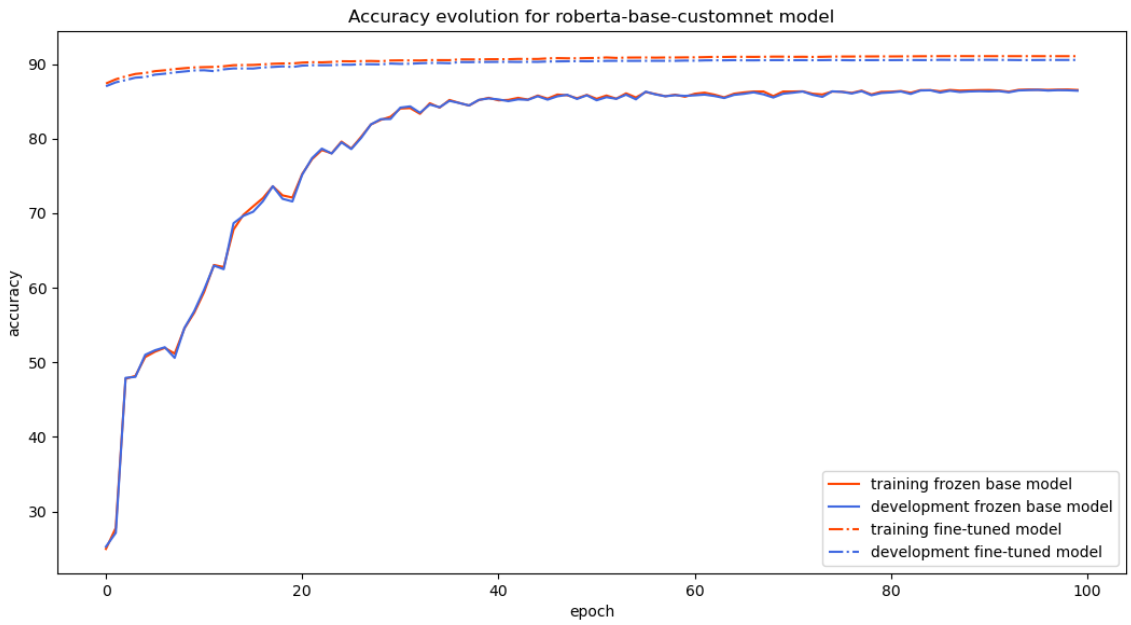

Figure 36: Dev accuracy evolution for RoBERTa CustomNet model

4.10 RoBERTa ConvNet

This is the same RoBERTa model described previously with convolutional, fully connected and dropout layers added after getting and reshaping the last hidden state from the base model to an image configuration. The following tables of results have been obtained for the RoBERTa ConvNet model.

		Predicted values				
	12000 samples	**World**	**Sports**	**Business**	**Sci/Tech**	*Macro Recall*
Actual values	**World**	1240	111	143	87	88.63%
	Sports	14	2913	8	4	99.12%
	Business	86	11	2656	277	87.66%
	Sci/Tech	95	15	182	2741	90.37%
	Macro Precision	93.16%	95.51%	88.86%	88.16%	

Table 32: Confusion matrix of RoBERTa ConvNet on development data

Again, the aim was to (unsuccessfully) improve the results obtained with the RoBERTa model by transforming an NLP problem into a Computer Vision one. The results are a little better than those obtained by the previous proposal, however, it is not enough. This model is placed in the third position, behind RoBERTa and BERT respectively, but adding a complex structure of reshapes and convolutions makes the whole process much more complex and computationally expensive.

We should mention the title of the paper that gave rise to the transformer-based models: "Attention Is All You Need", which, quite rightly, shows that no matter how many extra layers or mechanisms we add, we will only get worse results compared to those obtained by purely transformer-based models.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
91.39%	91.42%	91.44%	91.43%

Table 33: Average RoBERTa ConvNet macro-metrics on development data

The table below shows the rate of True Positives, True Negatives, False Positives and False Negatives for each class in the dataset.

12000 samples	True Positive (TP)	True Negative (TN)	False Negative (FN)	False Positive (FP)
World	2657	8807	341	195
Sports	2913	8924	26	137
Business	2656	8637	374	333
Sci/Tech	2741	8599	292	368

Table 34: Detailed confusion matrix for RoBERTa ConvNet with development data for every single class.

As in the original version, the evolution of accuracy in the development set almost mirrors that of the training set, with no clear evidence of overfitting.

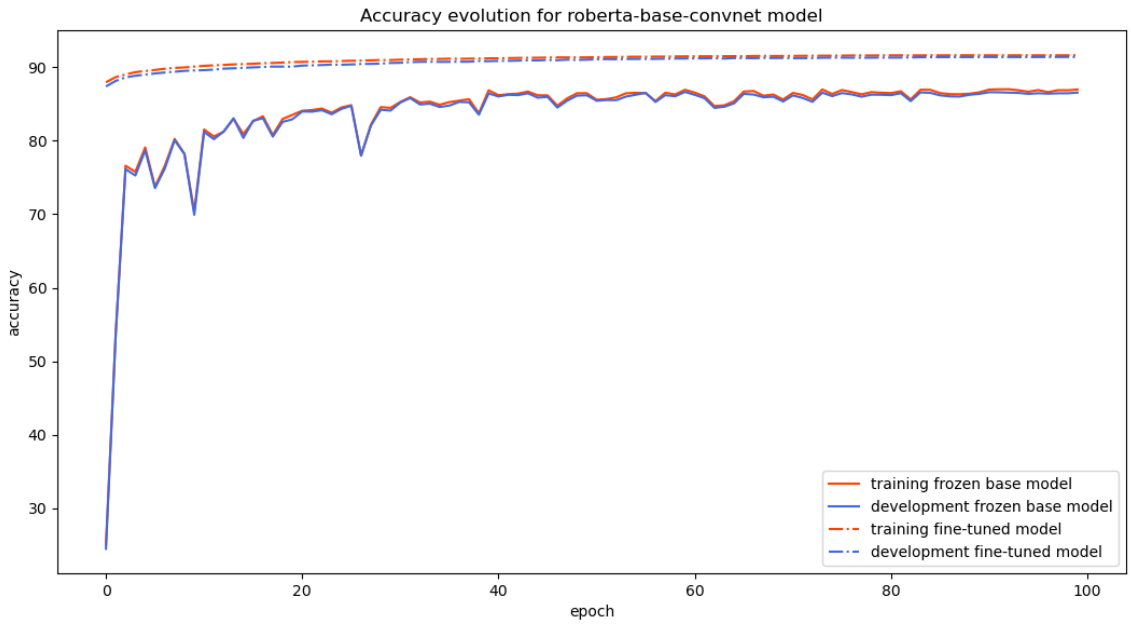

Figure 37: Dev accuracy evolution for RoBERTa ConvNet model

4.11 Development set results summary

100 Epochs		
Model	**Type**	**Accuracy**
LSTM	Standard	86.31%
LSTM II	Bidirectional	89.57%
bert-base-cased	Frozen	74.59%
	Fine-tuned	91.80%
bert-base-uncased	Frozen	78.89%
	Fine-tuned	88.33%
distilbert-base-uncased	Frozen	85.32%
	Fine-tuned	87.90%
roberta-base	Frozen	82.49%
	Fine-tuned	**92.07%**
xlm-roberta-base	Frozen	82.71%
	Fine-tuned	90.8%
gpt2	Frozen	50.02%
	Fine-tuned	50.66%
custom-net*	Frozen	86.52%
	Fine-tuned	90.55%
custom-convnet*	Frozen	86.63%
	Fine-tuned	91.39%

Table 35: Model results overview for development set

*Using roberta-base as a starting point

Having analysed all the selected models, the results obtained in the development set for the best version of each one can be seen in the table above. Here again, the best model obtained is the RoBERTa base with 92.07% accuracy. Followed by the original BERT model, both are characterised by their stability in training and metrics.

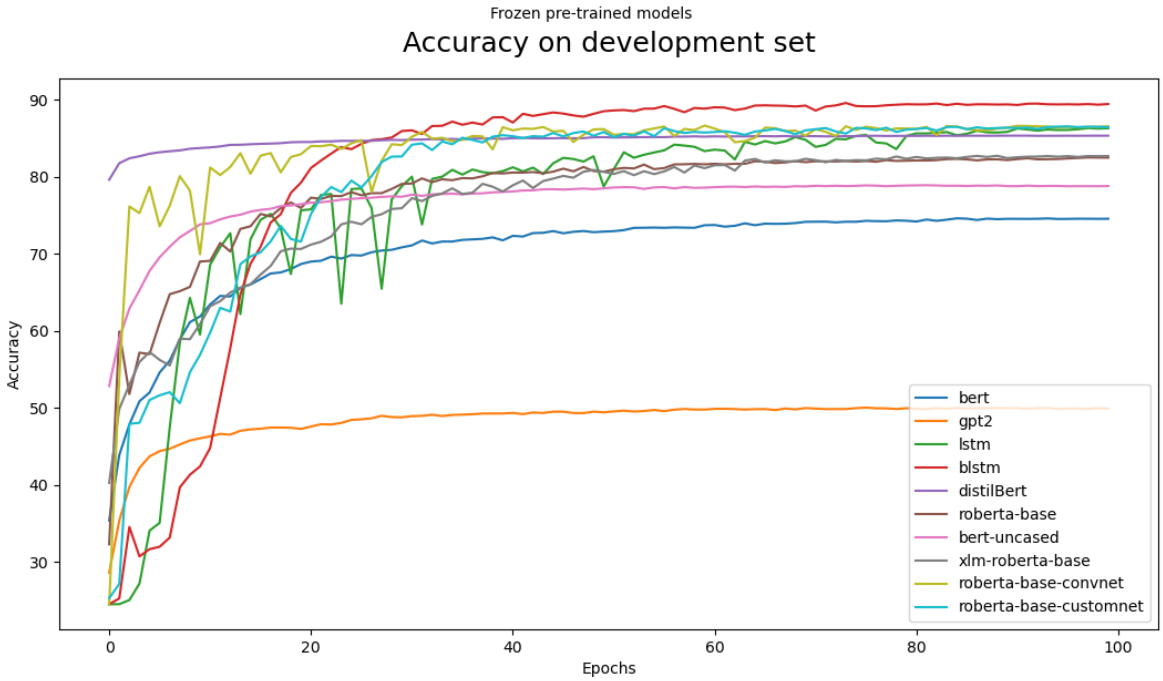

Figure 38: Overall accuracy evolution on the development set

However, there are models that, despite being worse in their metrics, can be very valuable in other contexts, as not everything is about accuracy. Often the speed of execution and resource consumption is even much more important. This is the case with bi-directional LSTM networks, for example. After 100 iterations achieve stabilisation near the 90% threshold with a forward cost considerably lower than provided by the transformer-based models.

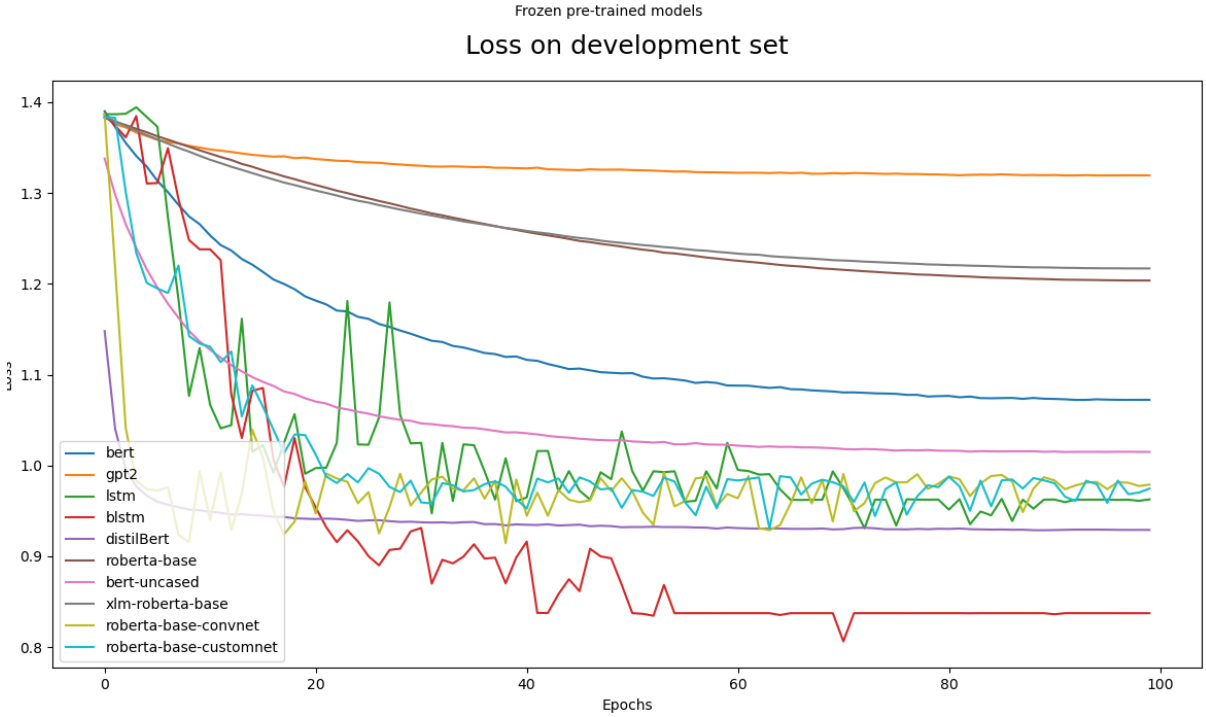

Figure 39: Overall loss evolution on the development set

Furthermore, as seen in the figure above, the bidirectional LSTM network can achieve the lowest values for the loss function in the first stage of training. It is only followed (fairly well behind) by the DistilBERT model, which although it is not as accurate as the bidirectional LSTM network, it is important to take into consideration that it can stabilise at around 85% accuracy in a little less than 20 epochs. This makes it an ideal model when training time is limited.

We can also observe that GPT-2 obtains disappointing results due to what we mentioned earlier. GPT-2 is decoder-only [16], which means that it omits the encoder part and that is critical for text classification. Due to its low performance, it has been decided to discard this model for the following figure. Otherwise, GPT-2 would interfere with the visualisation of the graphs, making the evolution of the rest of the models harder to see.

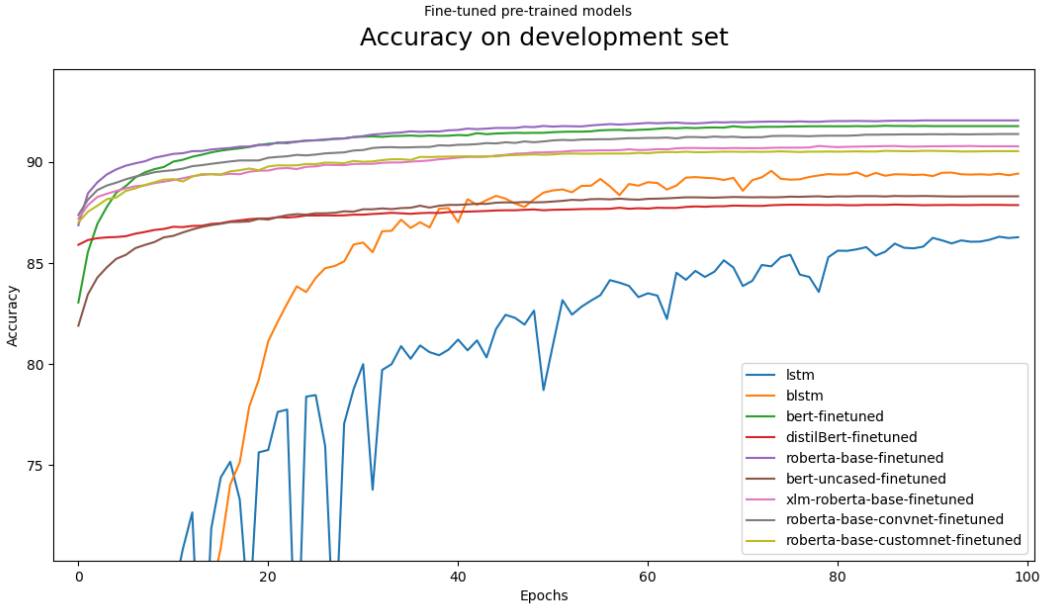

Figure 40: Fine-tuned model's overall accuracy evolution on the development set.

As we can see, the extra 100 epochs of fine-tuning work very well for the Transformer-based models. Since the top five models shown in the figure above are of this type. Surprisingly, the models offering the best performance for the loss function at this stage are respectively: BERT base, bidirectional LSTM networks and XLM RoBERTa. This means that while RoBERTa classifies better, it makes some huge mistakes in certain predictions.

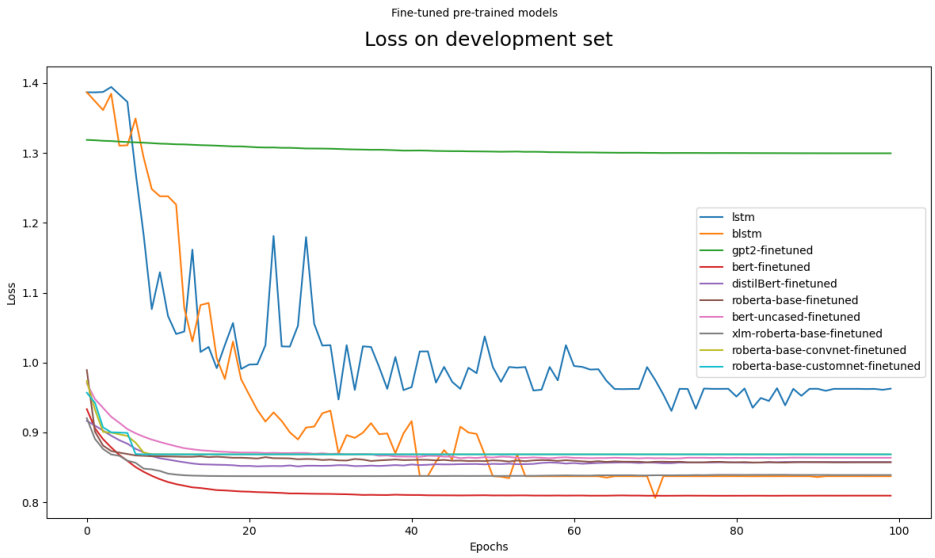

Figure 41: Fine-tuned models overall loss evolution on the development set

Chapter V - Results

Unlike the previous chapter, this section shows the results obtained by each model in the test set. That means that during training, the development set has been used to check the model performance every time it improved. This undoubtedly causes an inherent bias, so it is imperative to verify that the model does indeed learn by feeding it with samples that have never been processed and checking its performance. As is evident, all models show a very slight drop in their accuracy score, although it should be highlighted that this drop is in any case fairly acceptable.

5.1 LSTM

For the LSTM networks, we observe an overall accuracy increase by hundredths, so we can effectively assume that the results obtained with the test set are representative of those obtained during training. Surprisingly, in macro terms, the score for the LSTM model improves very slightly with samples never seen before.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
86.34%	86.34%	86.34%	86.34%

Table 36: Average LSTM macro-metrics on test data

Looking at the confusion matrix below, it is interesting to note that for the test set, there are some specific categories that are slightly better than those of the development set, such as World, Business and Sci/Tech. With slight drops for the sports category.

		Predicted values				
	7600 samples	**World**	**Sports**	**Business**	**Sci/Tech**	*Macro Recall*
Actual values	**World**	1628	75	104	93	85.68%
	Sports	45	1788	16	51	94.11%
	Business	99	17	1560	224	82.11%
	Sci/Tech	107	52	155	1586	83.47%
	Macro Precision	86.64%	92.55%	85.01%	81.17%	

Table 37: Confusion matrix of LSTM on test data

5.2 Bidirectional LSTM

Following the trend of the results obtained with the LSTM model, its bi-directional version improves the overall score very slightly in macro terms with samples that have never been seen previously.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
89.71%	89.72%	89.71%	89.72%

Table 38: Average Bidirectional LSTM macro-metrics on test data

Looking at the confusion matrix below, it is interesting to note that for the test set, there are some specific categories that are very slightly better than those of the development set, such as Sports and Business. With slight drops in other categories.

		Predicted values				
	7600 samples	**World**	**Sports**	**Business**	**Sci/Tech**	*Macro Recall*
Actual values	**World**	1677	69	96	58	88.26%
	Sports	27	1827	23	23	96.16%
	Business	75	14	1640	171	86.32%
	Sci/Tech	55	21	150	1674	88.11%
	Macro Precision	91.44%	94.61%	85.91%	86.92%	

Table 39: Confusion matrix of Bidirectional LSTM on test data

5.3 BERT

In the case of the BERT model, there is an expected not very significant decrease of a few tenths. So we can assume that the results obtained with the test set are representative of those obtained during training.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
91.50%	91.51%	91.50%	91.51%

Table 40: Average BERT macro-metrics on test data

Looking at the confusion matrix below, we can appreciate slight improvements in some specific categories (sports and sci/tech) if we look at the precision metric. However, this is not the case for recall, which generally decreases for all categories with the exception of the World category, which slightly improves.

		Predicted values				
	7600 samples	**World**	**Sports**	**Business**	**Sci/Tech**	*Macro Recall*
Actual values	**World**	1702	60	81	57	89.58%
	Sports	8	1875	12	5	98.68%
	Business	60	9	1649	182	86.79%
	Sci/Tech	48	11	113	1728	90.95%
	Macro Precision	93.62%	95.91%	88.89%	87.63%	

Table 41: Confusion matrix of BERT on test data

5.4 BERT uncased

BERT uncased model generally exhibits a drop of approximately 0.5% in all its macro metrics. So even though it cannot be said to be unrepresentative of the model.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
87.96%	87.93%	87.96%	87.95%

Table 42: Average BERT uncased macro-metrics on test data

Looking at the confusion matrix below, we can see slight decreases in all categories (except sports) if we look at the precision metric. However, this is not the case for recall, which increases slightly for Sports and World and decreases for the rest.

		Predicted values				
	7600 samples	World	Sports	Business	Sci/Tech	*Macro Recall*
Actual values	World	1654	77	107	62	87.05%
	Sports	56	1815	12	17	95.53%
	Business	93	20	1571	216	82.68%
	Sci/Tech	64	28	163	1645	86.58%
	Macro Precision	88.59%	93.56%	84.78%	84.79%	

Table 43: Confusion matrix of BERT uncased on test data

5.5 DistilBERT

DistilBERT holds its own while maintaining generally the same macro metrics with a very slight drop of approximately 0.15%.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
87.75%	87.75%	87.75%	87.75%

Table 44: Average DistilBERT macro-metrics on test data

Looking at the confusion matrix below, we can see increases in some categories like Sports or World if we look at the precision metric. For recall, it maintains almost the same behaviour.

		Predicted values				
	7600 samples	**World**	**Sports**	**Business**	**Sci/Tech**	*Macro Recall*
Actual values	**World**	1638	81	120	61	86.21%
	Sports	52	1814	19	15	95.47%
	Business	69	25	1597	209	84.05%
	Sci/Tech	59	36	185	1620	85.26%
	Macro Precision	90.10%	92.74%	83.13%	85.04%	

Table 45: Confusion matrix of DistilBERT on test data

5.6 RoBERTa

In the case of the RoBERTa model, we obtain an expected not very significant decrease of approximately 0.16%. So we can assume that the results obtained with the test set are representative of those obtained during training.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
91.91%	91.93%	91.91%	91.92%

Table 46: Average RoBERTa macro-metrics on test data

Looking at the confusion matrix below, we can appreciate slight improvements in some specific categories (World and Sports) if we look at the precision metric. However, this is not the case for recall, which generally decreases for all categories with the exception of the World category, which slightly improves.

		Predicted values				
	7600 samples	**World**	**Sports**	**Business**	**Sci/Tech**	*Macro Recall*
Actual values	**World**	1706	58	80	56	89.79%
	Sports	8	1879	9	4	98.89%
	Business	52	9	1668	171	87.79%
	Sci/Tech	43	10	115	1732	91.16%
	Macro Precision	94.31%	96.06%	89.10%	88.23%	

Table 47: Confusion matrix of RoBERTa on test data

5.7 XLM-RoBERTa

DistilBERT holds its own while maintaining generally the same macro metrics with a very slight drop of approximately 0.15%.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
90.67%	90.70%	90.67%	90.68%

Table 48: Average XLM-RoBERTa macro-metrics on test data

Looking at the confusion matrix below, we can that it still maintains almost the same behaviour.

		Predicted values				
	7600 samples	World	Sports	Business	Sci/Tech	*Macro Recall*
Actual values	World	1676	61	108	55	88.21%
	Sports	12	1866	17	5	98.21%
	Business	53	18	1648	181	86.74%
	Sci/Tech	51	12	136	1701	89.53%
	Macro Precision	93.53%	95.35%	86.33%	87.59%	

Table 49: Confusion matrix of XLM-RoBERTa on test data

5.8 GPT-2

This model has been discarded due to its proven ineffectiveness for this type of task, as discussed in chapter IV.

5.9 RoBERTa CustomNet

For the first proposed contribution, this model improves the overall score very slightly in macro metrics with samples never seen before.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
90.95%	91.00%	90.95%	90.97%

Table 50: Average RoBERTa CustomNet macro-metrics on test data

This model slightly improve, in general, all metrics for each class.

		Predicted values				
	7600 samples	**World**	**Sports**	**Business**	**Sci/Tech**	*Macro Recall*
Actual values	**World**	1680	61	107	52	88.42%
	Sports	9	1875	10	6	98.68%
	Business	39	7	1690	164	88.95%
	Sci/Tech	60	13	160	1667	87.74%
	Macro Precision	93.96%	95.86%	85.92%	88.25%	

Table 51: Confusion matrix of RoBERTa CustomNet on test data

5.10 RoBERTa ConvNet

The second proposed contribution generally exhibits a drop of approximately less than 0.1% in all its macro metrics. So it cannot be said to be unrepresentative of the model.

Avg. Accuracy	Avg. Macro Precision	Avg. Macro Recall	Avg. Macro F1 Score
91.37%	91.37%	91.37%	91.37%

Table 52: Average RoBERTa ConvNet macro-metrics on test data

As the original RoBERTa model, Looking at the confusion matrix below, we can appreciate slight improvements in some specific categories (World and Sports) if we look at the precision metric. However, this is not the case for recall, which generally decreases for all categories with the exception of the World category, which slightly improves.

		Predicted values				
	7600 samples	**World**	**Sports**	**Business**	**Sci/Tech**	*Macro Recall*
Actual values	**World**	1693	63	95	49	89.11%
	Sports	10	1879	8	3	98.89%
	Business	48	10	1661	181	87.42%
	Sci/Tech	57	12	120	1711	90.05%
	Macro Precision	93.64%	95.67%	88.16%	88.01%	

Table 53: Confusion matrix of RoBERTa ConvNet on test data

5.11 Test set results summary

100 Epochs		
Model	**Type**	**Accuracy**
LSTM	Standard	86.34%
LSTM II	Bidirectional	89.71%
bert-base-cased	Frozen	73.88%
	Fine-tuned	91.50%
bert-base-uncased	Frozen	78.05%
	Fine-tuned	87.96%
distilbert-base-uncased	Frozen	85.22%
	Fine-tuned	87.75%
roberta-base	Frozen	82.20%
	Fine-tuned	**91.91%**
xlm-roberta-base	Frozen	82.72%
	Fine-tuned	90.67%
gpt2	Frozen	49.78%
	Fine-tuned	50.70%
custom-net*	Frozen	86.30%
	Fine-tuned	90.95%
custom-convnet*	Frozen	86.91%
	Fine-tuned	91.37%

Table 54: Model results overview for test set

*Using roberta-base as a starting point

The behaviour of the models has not changed substantially when evaluated with previously unseen samples. In general, there is an expected slight drop in their performance, but nothing remarkable. Therefore, we can draw the same conclusions as mentioned in chapter IV.

In the figure below it can be seen the maximum accuracy obtained by each model in their first 100 epochs (in blue) and how the transformer-based models have improved (in green) during their fine-tuning stage. It should be noticed that all transformer-based models need a mandatory fine-tuning stage in order to even achieve the results of the bi-directional LSTM model.

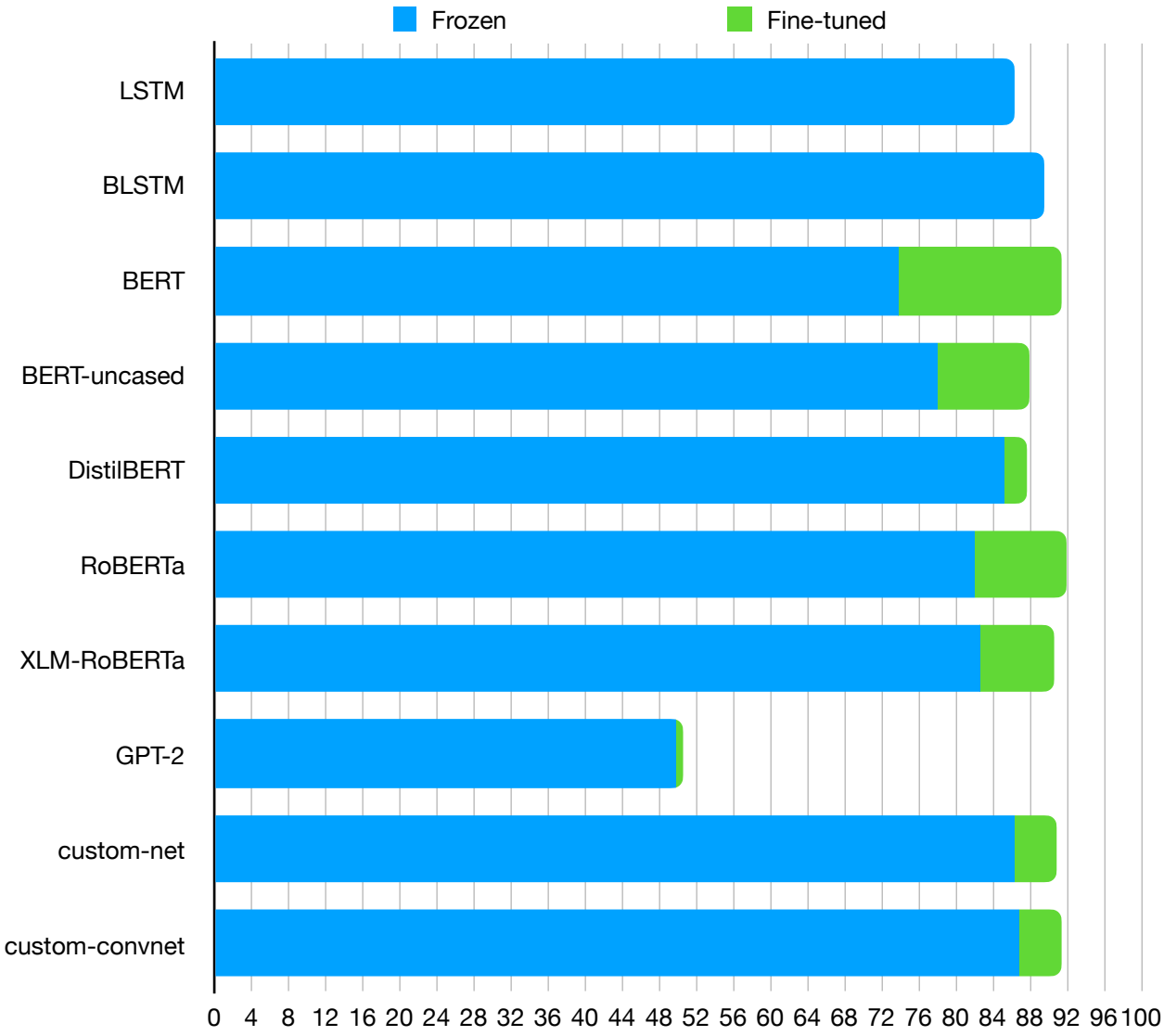

Figure 42: Accuracy bar-plot breakdown on test data

Chapter VI - Real-world model implementation

In addition to the previous work, a web interface has been developed for the RoBERTa model, which has given the best results. For this purpose, a docker environment has been created where two containers are hosted. The first is the PHP web server and the other has the model preloaded waiting for texts through POST requests.

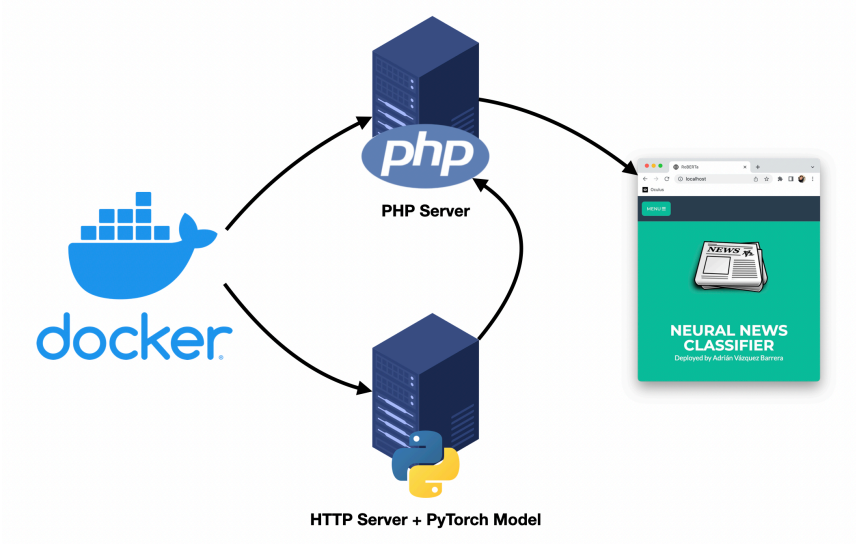

Figure 43: Deployment diagram

6.1 Docker

Docker is a set of platform-as-a-service products that use OS-level virtualisation to deliver software in packages called containers. This allows software solutions to be packaged together so the final production and development environments are virtually identical. This directly solves any incompatibility or dependency problems.

It has been opted to deploy two independent containers connected by a virtual network. The first one serves the API publicly through POST requests, while the other one simply consumes the API, presenting its functionality through a web interface.

6.2 Python HTTP server API

This container is responsible for serving the text classification API. It is the minimum necessary implementation that loads the model ready to evaluate strings. Finally, the HTTP library integrated with Python is used to enable the reception of texts through web requests for their evaluation and subsequent responses.

6.3 PHP server

This is a container running an Apache PHP server. It is designed to consume the API, dressing it up with graphical elements that make it user-friendly, combining HTML, CSS and JavaScript.

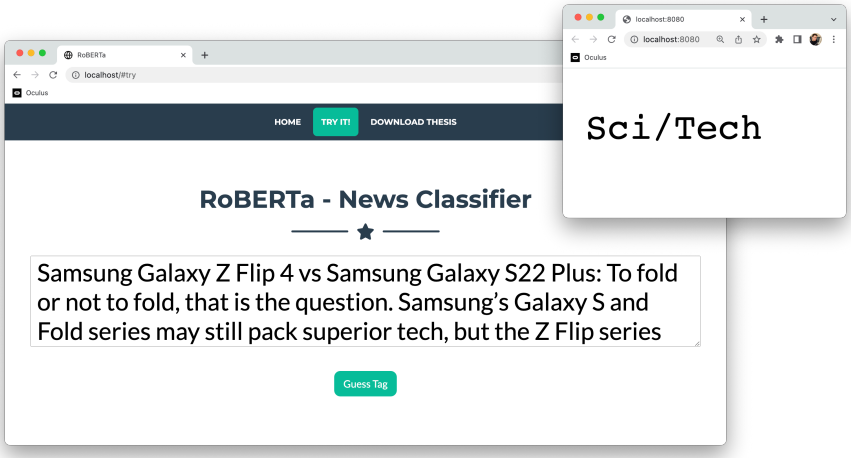

Figure 44: Operating example of the deployed system

Through the web interface by typing on the text box and then clicking the button, the system will provide the inferred label.

Chapter VII - Conclusions

All the models studied (except for GPT-2) reach the 85% accuracy threshold in the test set predictions. It should be considered that the transformer-based models have a hundred additional epochs of advantage to fine-tune pre-trained models.

Indisputably those models that have obtained the best results are BERT-based models. For the case of this task, the best one is the base RoBERTa model [24], followed by the original BERT and the proposed convolutional network based on the winning model. This simply supports the " Attention is all you need" paper [5] in which the authors suggest that learning the global dependencies between input and output can just be done with attention mechanisms.

The models proposed with the idea of improving the RoBERTa base model's performance for the text classification task have proved to be a fiasco. The best of the two models, RoBERTa ConvNet, only achieved the third position in the ranking. It is considered a failure because it adds complexity to the model which was supposed to improve.

Therefore, it would be appreciated which of the previous state-of-the-art models are capable of outperforming those transformer-based ones during the first hundred epochs. This is the case of bidirectional LSTM networks, which is undoubtedly the network which obtains the best results in the shortest time, nearing the 90% accuracy threshold in its first forty epochs.

If we have in mind that DistilBERT [22] is a model derived from BERT with several modifications that make it lighter and therefore less expensive but more limited in its results, it is worth highlighting that during the first phase of training, it is the model that starts with the best results in its first forty epochs. So, in case of need execution and training speed that can tolerate an error rate of around 20%, it might be an interesting option to consider. This advantage unfortunately does not hold in the fine-tuning process where other transformer-based models generally take the lead.

The multilingual [4] version of RoBERTa has obtained fairly decent results during the training. The fact of having a monolingual text corpus has been a handicap. It is perfectly feasible that if we would train this model with examples from different languages, in the worst scenario the performance could be equivalent to the ability to classify internationally with about a 10% error rate.

GPT-2 is a pre-trained model trained in a self-supervised way. It is therefore particularly good at answering questions, translating, generating and summarising texts, but has poor classification skills. This explains the terrible performance of this model, which is barely better than flipping a coin. GPT-2 does not have an encoder as the original transformer architecture does [16], as it is decoder only, there are no encoder attention blocks, so the encoder is equivalent to the encoder itself, except for the masking in the multi-head attention block. This explains its malfunctioning, as the most important part of classifying texts is the encoder.

BERT was undoubtedly a breakthrough in the use of machine learning for natural language processing. The fact that it is accessible and allows quick fine-tuning, enables a wide range of practical applications and finally, four years later the BERT model and its derivatives are leading [21] at least the text classification field.

In general, all models show a great facility to differentiate sports news. This may be due to the fact that political, technological and business news are nowadays quite intertwined. For example, by 2022 it is not surprising to observe political movements as a result of Elon Musk's declarations about his intention to buy the wildly popular bluebird social network.

The results obtained by other users using similar techniques, even though they have used different data splits, are shown below.

Model	Accuracy	Split	Epochs
Multinomial Naive Bayes	89.62%	Train-test split	-
BERT [18]	94%	Train-test split	4
LSTM	90.41%	Train-test split	10

Table 55: Results obtained by other users who used similar techniques with different data partitions.

The difference in results can be perfectly explained by the use of only two partitions, which adds more samples for training. The fact of saving the model that provides the best accuracy in the test also conditions it. The reduced number of epochs does not allow us to see the long-term evolution where the model stabilises.

While deploying the model it should be noted that, if we opt for a Transformer-based model, a CUDA-compatible GPU will be almost mandatory. This is because, even if we only have to forward evaluate a sample, these models have a large number of parameters which can increase the response time. It is also a fact that the machine used to deploy the RoBERTa model is a macOS machine with an ARM processor under docker, which implies (at least at the moment) several consecutive emulations, x86 architecture emulation from ARM running a Linux container with all of that running on the CPU.

This should encourage a reconsideration of the choice of a model for deployment, as the most accurate model does not always have to be the best. There is a trade-off between the hardware cost and the required performance for each model to be executed in a given time. So, depending on what we want, we will have to consider a balance between speed and accuracy.

Chapter VIII - Future work

The study of the transformer-based language models seen in this paper provides a basis for numerous further studies, ranging from the more evident ones, such as multi-label support, fake news detection, etc., to those that explore different topologies from the typical many-to-one, such as news summarisation. However, there is also a significant margin for improvement without necessarily changing the nature of the original task.

8.1 Multi-lingual datasets

Previous experiments have shown that multilingual models such as XLM-RoBERTa have only a 1.5% difference in the accuracy rate compared to the best monolingual models. This means that in the case of a multilingual corpus, we could obtain better results.

8.2 Large language models (LLM)

Some language models have recently been released to compete directly with GTP-3 [20], some of which have been published during the execution of this project and which due to time or lack of resources it has not been possible to implement in this study.

Examples of recently published LLMs include:

- OPT-175B[9] - Facebook, Inc.
- BLOOM[10] - Hugging face, Inc.
- PaLM[11] - Google, Inc.

[9] https://arxiv.org/abs/2205.01068

[10] https://huggingface.co/bigscience/bloom

[11] https://ai.googleblog.com/2022/04/pathways-language-model-palm-scaling-to.html

Acknowledgements

I would like to express my gratitude to my advisor Francisco Casacuberta Nolla, who has guided me throughout this project.

I would also like to thank my friends and family who have supported and offered me all the help I could need. Mostly for giving me that little push I needed during my first year of university studies. Thanks to them, I was introduced to computer science and later on, I discovered all that artificial intelligence has to offer.

I would like to extend my special thanks to all the teachers I have had since I entered the school system. From elementary school to the highest grades, every little bit counts in this journey that has spanned almost two decades. With special thanks to Walter Valentine Cole Rogers, who was my spoken English teacher last year. He strongly encouraged me to study for this master's degree and regrettably has recently passed away, I will never forget you.

Bibliography

[1] A. Bhavani & B. Santhosh Kumar (2021) A Review of State Art of Text Classification Algorithms. IEEE.

[2] Alex Graves, Santiago Fernández, Faustino Gomez & Jürgen Schmidhuber (2006) Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. IDSIA.

[3] Alex Sherstinsky (2018) Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. MIT. arXiv:1808.03314 [cs.LG]

[4] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer & Veselin Stoyanov (2019) Unsupervised Cross-lingual Representation Learning at Scale. Facebook AI. arXiv:1911.02116 [cs.CL]

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser & Illia Polosukhin (2017) Attention Is All You Need. arXiv:1706.03762 [cs.CL].

[6] Bijoyan Das & Sarit Chakraborty (2018) An Improved Text Sentiment Classification Model Using TF-IDF and Next Word Negation. arXiv:1806.06407 [cs.CL].

[7] Foram P. Shah, Vibha Patel (2016) A review on feature selection and feature extraction for text classification. IEEE.

[8] Graves, A. (2012). Connectionist Temporal Classification. In: Supervised Sequence Labelling with Recurrent Neural Networks. Studies in Computational Intelligence, vol 385. Springer, Berlin, Heidelberg.

[9] Haiyi Zhang, Di Li (2007) Naïve Bayes Text Classifier. IEEE.

[10] Haojin Hu, Mengfan Liao, Chao Zhang & Yanmei Jing (2020) Text classification based recurrent neural network. IEEE.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee & Kristina Toutanova (2018) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Google. arXiv:1810.04805 [cs.CL].

[12] Jeffrey Pennington, Richard Socher, & Christopher D. Manning. (2014). GloVe: Global Vectors for Word Representation. Computer Science Department, Stanford University.

[13] Jonathan J. Webster & Chunyu Kit (1992) Tokenization as the initial phase in NLP.

[14] Li, C., Zhan, G. & Li, Z. (2018) News Text Classification Based on Improved Bi-LSTM-CNN. 9th International Conference on Information Technology in Medicine and Education, Hangzhou.

[15] P.J. Werbos (1990) Back-propagation through time: what it does and how to do it. IEEE.

[16] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI.*

[17] Ralf C. Staudemeyer & Eric Rothstein Morris (2019) Understanding LSTM -- a tutorial into Long Short-Term Memory Recurrent Neural Networks. arXiv:1909.09586 [cs.NE].

[18] Samin Mohammadi & Mathieu Chapon (2020) Investigating the Performance of Fine-tuned Text Classification Models Based on Bert. IEEE.

[19] Selva Birunda, S. & Kanniga Devi, R. (2021). A Review on Word Embedding Techniques for Text Classification. In: Raj, J.S., Iliyasu, A.M., Bestak, R., Baig, Z.A. (eds) Innovative Data Communication Technologies and Application. Lecture Notes on Data Engineering and Communications Technologies, vol 59. Springer, Singapore.

[20] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo & Yusuke Iwasawa (2022) Large Language Models are Zero-Shot Reasoners. University of Tokyo. arXiv:2205.11916 [cs.CL].

[21] Tingyu Zhang & Ruixia Zhang (2021) Revealing the power of BERT for text sentiment classification. IEEE.

[22] Victor Sanh, Lysandre Debut, Julien Chaumond & Thomas Wolf (2019) DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. Hugging Face. arXiv:1910.01108 [cs.CL].

[23] Vidish Sharma, Aditya Bendapudi, Tarun Trehan, Ashutosh Sharma, Adwitiya Sinha (2020) Analysing Political Bias in Social Media. IEEE.

[24] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer & Veselin Stoyanov (2019) RoBERTa: A Robustly Optimized BERT Pretraining Approach. Facebook AI. arXiv:1907.11692 [cs.CL].

[25] Ziqiang Wang, Xu Qian (2008) Text Categorization Based on LDA and SVM. IEEE

[26] Ben F. (2015) Clickbait: The changing face of online journalism. BBC. BBC news.

[27] Elias O. & Delermando B. (2016) Automatic classification of journalistic documents on the Internet. Faculdade Católica Salesiana do Espírito Santo, Brazil.

[28] Sarah T. (2014) Will the printed newspaper be extinct by 2040?. Medium.