MASTER'S THESIS

Interuniversity Master in Statistics and Operations Research UPC-UB

Title: Attention Mechanisms in Deep Learning Models for Twitter Sentiment Analysis

Author: Elena Blanco González

Advisor: Ferrán Reverter Comes and Esteban Vegas Lozano

Department: Department of Genetics, Microbiology & Statistics.

University: University of Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

UNIVERSITAT DE BARCELONA

Master in Statistics and Operations Research

Attention Mechanisms in Deep Learning Models for Twitter Sentiment Analysis

Author:

Elena Blanco González

Advisors:

Ferrán Reverter Comes

Esteban Vegas Lozano

January, 2023

Abstract

Sentiment Analysis on social media such as Twitter can provide us with valuable information about the users' opinions. The singularities of these data lie in their short format and informal language. In the last years, Deep Learning models like Recurrent Neural Networks and Convolutional Neural Networks have been widely studied for this task reaching promising results when combined with word embedding mechanisms. In this master thesis, we go through the bases of Sentiment Analysis and Deep Neural Networks and then some Deep Learning models are presented. Pursuing improving the performance of these models, attention mechanisms like Self Attention of Transformer Encoder are presented and included in the models. The same dataset is used to train all the presented models in order to evaluate them and analyze the impact of including attention mechanisms on Deep Neural Networks in a Sentiment Analysis task.

Keywords

Sentiment Analysis, Deep Learning, Neural Network, Attention Mechanisms, Natural Learning Processing, Bidirectional Long-Short Term Memory

Acronyms

BiLSTM Bidirectional Long-Short Term Memory

BoW Bag of Words

CNN Convolutional Neural Network

DNN Deep Neural Network

LDA Latent Dirichlet Allocation

LSTM Long-Short Term Memory

MHAT Multi-Head Attention

NLP Natural Language Processing

RNN Recurrent Neural Network

SA Sentiment Analysis

SVM Support Vector Machine

Contents

Chapter 1

Introduction

A person wanting to buy a new product, a company worried about how people feel about its products, political parties concerned about the possibilities they have in the upcoming elections. There are all situations where someone is looking for people's opinions regarding an specific topic. Historically people asked friends and family before buying something in order to assure they were taking the best decision, companies did customer satisfaction surveys and political parties chose a representative part of society and ask them about the main points of their program. In all of these cases, the data has to be collected person by person and, after collecting the data, it should be manually classified in order to can extract conclusions.

Nowadays, individuals, companies, organizations and governments use the huge amount of data available in social media for decision making: reviews about the product you want to buy, tweets or blog posts about a social measure or a company, etc. Therefore, social media already avoids several manually data collection. The next step is to find a way to automate their classification in order to make all these data useful.

Moreover, neural networks, that were widely used for image classification, were adapted to Natural Language Processing (NLP) tasks by using word embeddings that allow to have vector representation of words where similar words are close on the vector space. Deep Neural Networks (DNN) have surpassed in terms of accuracy other traditional classification methods. Among all the DNN, Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) - specially with the presence of Long-Short Term Memory (LSTM) - are the most popular on text sentiment analysis.

In the last decade, it appears the concept of attention mechanisms and become a very powerful concept in deep learning improving the performance of neural networks. Attention mechanisms assign different weights do each token of the input allowing the neural network to focus on the more relevant parts for the specific task obtaining a better representation of it.

In this master thesis we will study the power of attention mechanisms when working with short texts by considering Twitter data. The first chapter is composed by an introduction of Sentiment Analysis (SA), DNN and GloVe embedding. Then, on Chapter 2, we present the enabled models divided in two categories: DNN architectures and Attention Mechanisms. On the third chapter we start presenting the data set that will be used to evaluate the five different models described on the second part of the chapter. Chapter 3 ends by comparing the different models and, in the final chapter, the reached conclusions and future work are exposed. There is also an appendix with the full Python code used.

Chapter 2

Deep Neural Networks for Sentiment Analysis

2.1 Sentiment Analysis

From some years to now, the amount of data available has significantly increased thanks to social media and blogs, where people write their opinion about several topics or products. Sentiment Analysis is the analysis of a text with the aim of obtaining people's opinion regarding an specific topic. For instance, SA provides an efficient way to determine if the expression in a text is positive or negative.

The increase of available data together with the reduction of computational costs converts SA in one of the most active research areas in Natural Language Processing. It has been spread form the field of computer science to a wide range of other disciplines more related with social science such as marketing, finances, politics or communications. SA has multiple applications on real world where knowing people's opinion helps organizations and companies to make better decisions by analyzing how people feel in a macro scale about a product, a service or even a brand.

Statistical methods such as Support Vector Machines (SVM), Latent Dirichlet Allocation (LDA) or Naïve Bayes have been used in text classification tasks. However, these methods present two main inconveniences first, they should be trained in a high-dimensional feature space, what decreases the performance of the model , and also the feature engineering process requires a lot of time and work [4].

In order to overcome this limitations, in the last years the community started to use word embedding that transform text into matrices considering the lexical relationships between words. By doing this, we can start using Deep Neural Networks for Natural Language tasks such as sentiment analysis.

2.2 Deep Neural Networks

The structure of DNN has been inspired by the structure of the human brain. DNN consist on a set of units called neurons organized in layers that work unison. This type of networks can learn to perform tasks by adjusting the connection weights between neurons.

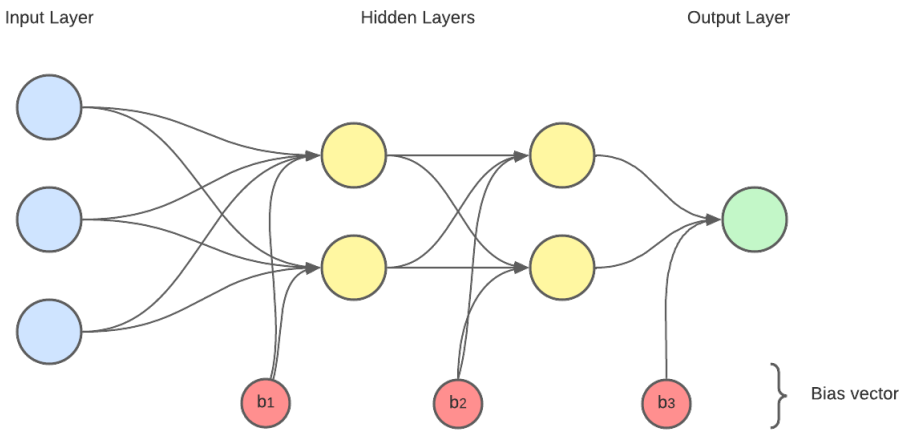


Figure 2.1: DNN Structure

Figure 2.1 shows the standard structure of a DNN. It is composed by three types of layers: one input layer, several hidden layers and an output layer. The values in the input layer denote the input data supplied to the network while the hidden and output layers are composed by neurons.

The flow of information between neurons is determining by the weights w and fitting this weights allows the network to learn features from the data. In each layer, each neuron takes its input x from the previous layer and it calculates an output value by applying an activation function g to the weighted sum of inputs and the bias b. Mathematically it is denoted as

$$g(W^t x) = g(\sum_i W_i x_i + b)$$

Activation functions are usually non-linear and the most common are *sigmoid*, hyperbolic tangent (*tanh*) and Rectified linear unit ($ReLU$).

$$sigmoid(W^t x) = \frac{1}{1 + e^{-W^t x}}$$

$$tanh(W^t x) = \frac{e^{W^t x} - e^{-W^t x}}{e^{W^t x} + e^{-W^t x}}$$

$$ReLU(W^t x) = max(0, W^t x)$$

The choice of the activation function in the output layer depends on the specific task that the network is performing. In the case of considering a classification task with more than two categories, the activation function should be the *softmax* function, similar to *sigmoid* function but adequate to handing multi-class problems, It turns a vector of K real values into a vector of K real values between 0 and 1 that sum 1 such that they can be interpreted as the probability to belong to each of the K classes. Generally this function is only used in the output layer and it is defined as follows.

$$\sigma(Z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \qquad \text{for } i = 1, ..., K$$

Neural networks are often trained with optimization techniques that need a loss function in order to estimate the model error. Depending on the learning task the loss function will be log-likelihood or sum of squares and the network parameters are optimized with the output of the loss function employing different optimization techniques. In this thesis, as we are considered a big data set and the models have a lot of parameters, we will be working with Adam optimization algorithm that uses Momentum and Adaptive Learning Rates to converge faster.

The most popular types of DNNs in tasks related to text processing are Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). The cause of this popularity is due to the fact that CNNs can of learning local patterns

and RNNs have the ability to analyze sequential data. In the next sections, RNNs and CNNs models will be considered. There will also be contemplated some hybrid models that combine both of them.

The next table summarizes some of the publications of the last five using RNNs and CNNs for sentiment analysis:

Year	Study	Research	Methods
2017	A. Hassan et al. [10]	Sentiment analysis on short texts	CNN, LSTM
2018	J. Qian et al. [14]	Sentiment analysis on weather-related tweets	DNN, CNN
2019	A. S. M. Alharbi et al. [2]	Twitter sentiment analysis	CNN
2019	J. Xie et al. [17]	Sentiment analysis on short texts	Self Attention based BiLSTM
2021	H. S. Sharaf Al-deen et al. [4]	Sentiment analysis on short texts	CNN, BiLSTM, Multi-Head Attention

2.3 Word embedding: GloVe

In order to be able to work with text data in DNN, texts need to be converted somehow to vectors. Traditionally, it has been popular the use of bag-of-words representation (BoW) that, given a dataset, it forms a "bag" with all the words appearing and then each entrance (sentence, document, tweet, etc) is transformed into a vector of length the number of words in the bag that shows how many times each word appears.

Imagine that we have the following 3 sentences:

- 'He is my young brother'
- 'He is a mechanic'
- 'His young brother is as tall as him '

The sentences have a total of 17 words that can be summarized in a bag of 11 words: 'He' , 'is' , 'my', 'young', 'brother', 'a', 'mechanic', 'his' , 'as', 'tall', 'him'. Therefore each sentence will correspond with a vector of length 10, $x = (x_1, x_2, ..., x_{10})$ with x_i the number of times that the word in the ith position of the bag of words is the sentence:

- 'He is my young brother' = (1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0)
- 'He is a mechanic' = (1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0)

- 'His young brother is as tall as him' = (0, 1, 0, 1, 1, 0, 0, 1, 2, 1, 1)

This representation has some inconveniences as the length of the vector corresponds to the vocabulary size so, when having a big data base like several books, the vectors will be huge and with a lot of 0's requiring a lot of memory and computational resources. Moreover, it does not take into account context and sentences with a very similar meaning can have really different representations.

Word embeddings overcome these problems as they try to map human language into a geometric space, therefore, words with similar meanings will be close in the word-embedding space. As we can see on Figure 2.2 the distance between the representations of 'brother' and 'sister' is similar to the one between 'nephew' and 'niece' or 'aunt' and 'uncle' as, in human language, the difference between these words is always the same one: the gender. There are several embedding algorithms: word2vec, ELMO, BERT, GloVe, etc.

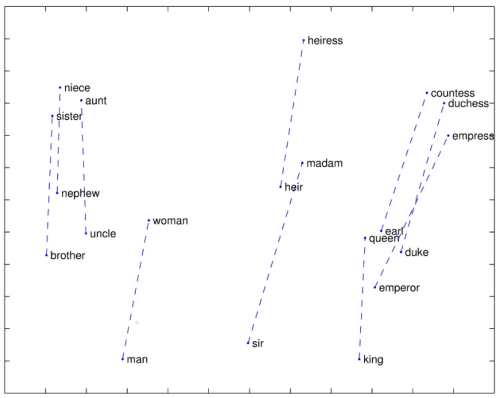


Figure 2.2: Distance of words in the word-embedding space [8]

In this thesis all the models considered are using a pre-trained GloVe embedding that has been trained on Twitter data [8].

GloVe model [13] is based on factorizing a matrix of word co-occurrences statistics.

Given a set of V words, the co-occurrence matrix X will be a matrix where X_{ij} denotes how many times the word j occurs in the context of word i. The probability of seeing this two words together is calculated by dividing the number of times i and k appear together divided by the number of times that the word i appears in the set of words: $P_{ik} = X_{ik}/X_i$. Therefore given the three words i, j and k (this third one is called *probe word*) if i and j are both similar or both unrelated to the

probe word k, P_{ik}/P_{jk} will be close to one. If i is similar to k but j is not, P_{ik}/P_{jk} will be a high number greater than 1 and in the opposite case - j similar to k and i different from k- the value will be very small.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Figure 2.3: Co-occurrence probabilities [13]

Figure 2.3 presents the co-occurrence probabilities for target words *ice* and *steam* with four other words: *solid*, *gas*, *water* and *fashion*.

Chapter 3

Enabled Models

After a brief description of Sentiment Analysis and the intuition on Deep Neural Networks, this section will be composed by two main parts. In the first part we consider different DNN architectures and in the second part attention mechanisms are presented.

3.1 DNN architectures

3.1.1 Convolutional Neural Networks

Convolutional Neural Networks are able to capture local features of data. The were initially used in the field of computer vision but they are widely used in other fields as speech recognition, text mining and sentiment analysis.

In order to classify a tweet according to its sentiment, CNN take as an input the matrix provided by the embedding layer and outputs the probability of a tweet to belong to each class.

Convolutional Neural Networks are composed by three type of layers: Convolutional, Pooling and Dense or Fully Connected. Pooling layers are optional but the other ones are always in the network as the convolutional layer is the one that captures the local features and dense layer outputs the probabilities of belonging to each class.

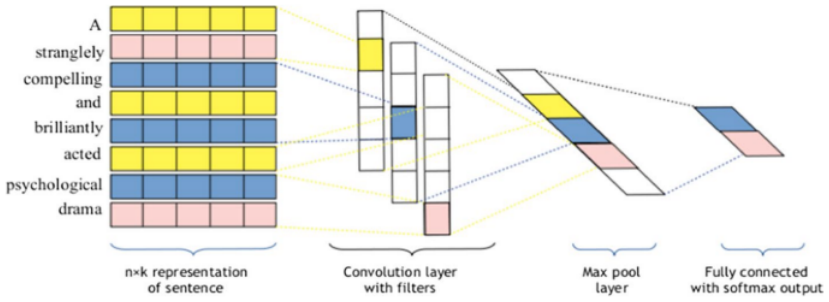


Figure 3.1: CNN architecture for sentiment classification [16]

Figure 3.1 represents the architecture of a CNN in the case of sentiment analysis. As can be seen, each row represents the k-dimensional word embedding of each word. Hence, if the length of each sentence is n, a matrix of dimension $n \times k$ is provided as input of the network.

Convolutional layers are the main component of a CNN architecture and they are composed by filters that extract features of the date and whose parameters should be learned. These filters f are applied over a window of k terms to generate a convoluted feature c_i:

$$c_i = f(x_{i:[i+k-1]}) + b$$

were b is the bias and f the activation function. Each filter is applied to all possible window of words in the sequence to generate the feature map.

The **Pooling Layer** applies some operation over the regions in the input feature map and extracts some representative value for each of the analyzed regions. By doing this, pooling layers increases the CNN's robustness to avoid noise and distortions. Contrary to convolutional layers, the function in pooling layers is fixed. Usually what they do is to calculate the average of the features values or the maximum, determining the essential feature of the map.

Even if **Dense Layers** can also be on the inside of the neural network, the output layer is always a **Dense Layer**. Dense layers perform the classification task. In a dense layer, each of the outputs of the previous layer is connected with to each neuron in the layer, which implies that each output dimension depends on each input dimension.

3.1.2 BiLSTM

Recurrent Neural Networks are a type of neural networks that have the form of a chain with a module that is repeated in such a way that for each step, the output is generated based on the output of the previous step and the input of the current one.

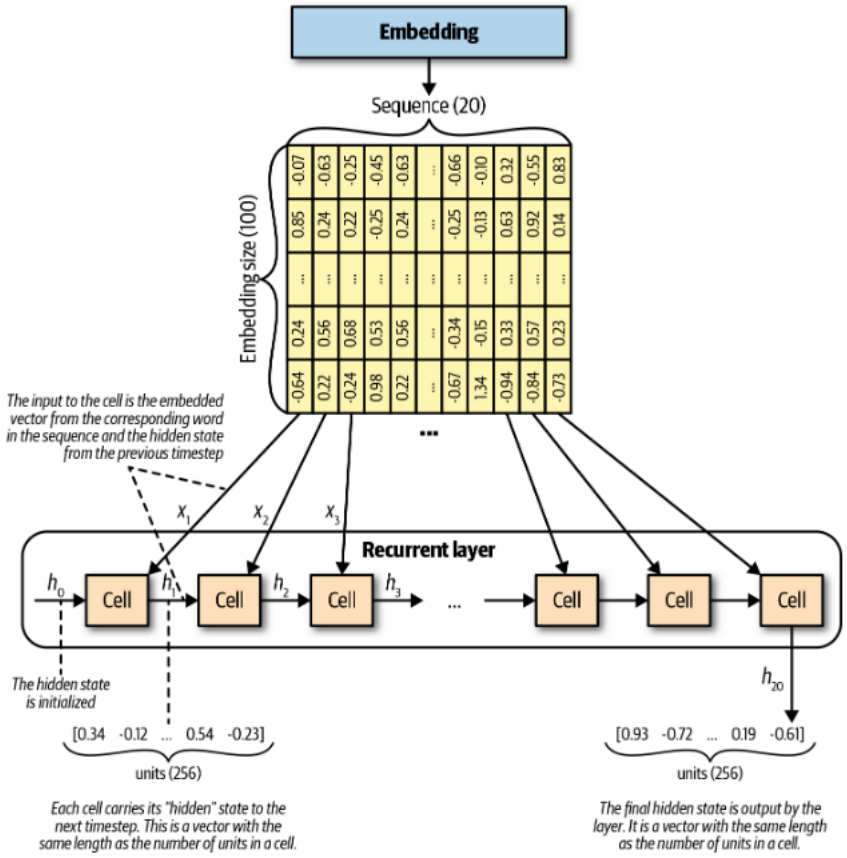


Figure 3.2: Long-Short Term Memory network [7]

Long-Short Term Memory is an special type of RNN whose module has a more complex structure than standard RNNs. LSTM are composed by a chain of recurrent memory units as shown on Figure 3.2. Each of the cells has four components: a memory cell and three gates (forget, input and output). These components interact among them such that the cell records information and the gates control

the flow from cell to cell as shown in Figure 3.3 [1].

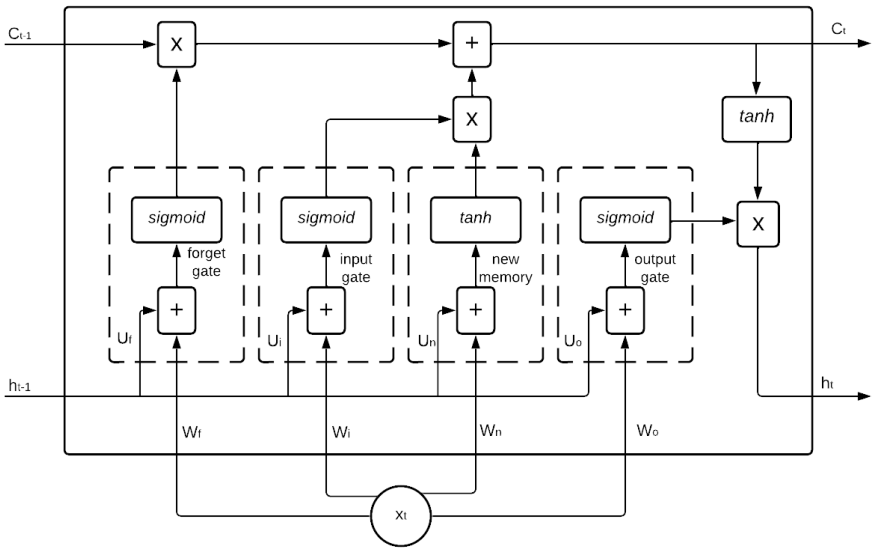


Figure 3.3: LSTM unit

Given the input $x = (x_1, x_2, ..., x_n)$ LSTM will generate the hidden vectors $h = (h_1, h_2, ..., h_n)$, being n the length of the sentence. The dimension of each vector x_i is the dimension of the embedding and the dimension of h_i coincides with the number of units in a cell. W and U are the weight matrices and b the biases of LSTM cell during training. The procedure is the following:

First, the **forget gate** decides what information discard for the cell state by using sigmoid function:

$$f_t = \sigma(W^f x_t + U^f h_{t-1} + b_f)$$

f_t will be a value between 0 and 1 where 0 will mean completely forget and 1, completely keep.

Then, the **input gate** will decide the new information to store in the cell. In order to do that, first it will calculate

$$i_t = \sigma(W^i x_t + U^i h_{t-1} + b_i)$$

[1]The image is an adaptation of an schema of L. Zhang et al. [18]

and create new candidate values to be added in the cell state $\tilde{c}_t$.

$$\tilde{c}_t = tanh(W^c x_t + U^c h_{t-1} + b_c)$$

Update the old cell state c_{t-1} into the new one being $\odot$ the element wise multiplication.

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

Finally the output is decided based on the cell state. The **output gate** is calculated using the formula

$$o_t = \sigma(W^o X_t + U^o h_{t-1} + b_o)$$

that decides which parts of the cell state to output and then, multiplying it by the cell state, only some parts of the output are shown on the hidden state h_t

$$h_t = o_t * tanh(c_t)$$

With the procedure described above, the outputs are calculated taking into account the previous context. Bidirectional LSTM (BiLSTM) includes both the previous and future context.

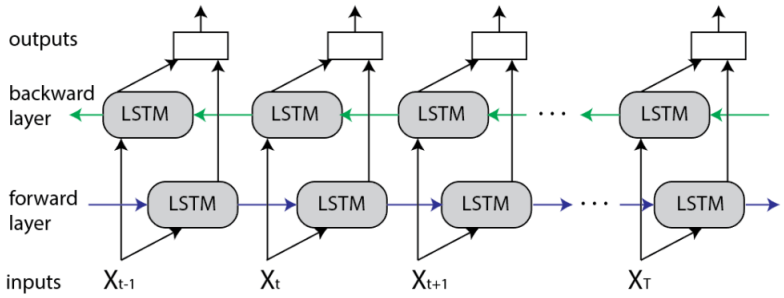


Figure 3.4: BiLSTM structure [5]

BiLSTM is represented on Figure 3.4. It is composed by two independent LSTM, one of them that goes forward thorough the input and anther one backwards that will obtain two different hidden layers: $\overrightarrow{h_t}$ for the forward LSTM and $\overleftarrow{h_t}$ for the backwards one. The final hidden vector h_t of the BiLSTM is obtained concatenating $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ as $h_t = [\overrightarrow{h_t}, \overleftarrow{h_t}]$.

In order to perform classification, the final hidden layer is the input of the dense layer.

3.2 Attention Mechanisms

The Attention Model was fist used in machine translation tasks but, over the years, it has become very popular in many other fields as NLP, statistical learning, speech recognition, computer vision or sentiment classification. The application of attention mechanisms on NLP has been one of the greatest advances of the last decade, specially after the publication of "Attention is All you Need" by the Google's machine translation team in 2017 [15].

In the case of sentiment analysis, given a tweet, not all the words contribute the same to the context of sentiment polarity. Attention mechanisms, try to imitate humans brain behaviour and allow the model to learn the parts of the input with a higher relevance for the specific task they are performing while forgetting the rest.

3.2.1 Self-Attention

Self-Attention focus on modulating a word representation by using the representation of related words in the sentence, therefore, self-attention is a context aware kind of attention.

The mechanism calculate an attention value for each input and then it outputs a context c, a weighted sum of the inputs according the relevance of each word based on the attention. First, for every word in the sentence, the algorithm computes the attention score based on every other word in the same sentence. In order to do that, it uses the dot product between two vectors (word representations) as a measure of the strength of their relation. Then a scaling function and softmax will be applied to the calculate dot product. Lastly, the weighted sum of all word vectors in the sentence is calculated to obtain the context.

Specifically, when applying an attention layer after BiLSTM, the network will get the hidden vector h_t produced by BiLSTM as an input to obtain the context information c for each sequence. In order to do that, first the hidden representation is transformed into another hidden representation u_t

$$u_t = tanh(W^w h_t + b_w)$$

where W^w is a weight matrix and b_w a bias vector. Later, the attention value is

computed considering u_t and a word-level context vector u_w that helps to distinguish the importance of different words in the sentence.

$$a_t = \frac{\exp(u_t^T u_w)}{\sum_t \exp(u_t^T u_w)} = softmax(u_t^T u_w)$$

The sum of the attention values of a sequence must be 1 and the higher the value, the more relevance the word is in the context of sentiment polarity. Finally the context value is calculated

$$c = \sum_t a_t h_t$$

W^w, b_w and u_w are randomly initialized and then they are adjusted during the training process. Figure 3.5 shows an schema of the process.

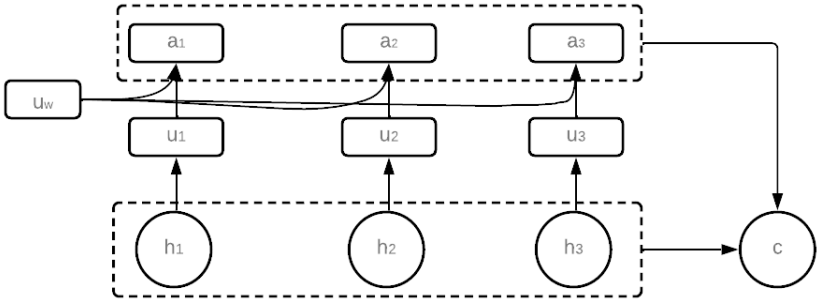


Figure 3.5: Attention Mechanism

3.2.2 Transformers

Transformers were introduced in the paper "Attention is all you need" [15] and they were originally developed for machine translation. They consist on two parts: an *encoder* that process the input sequence and a *decoder* that generates the new sentence in the new language. However, the encoder part can also be used for text classification because what it does is to create a new representation of the sentence giving more importance to the most relevant parts.

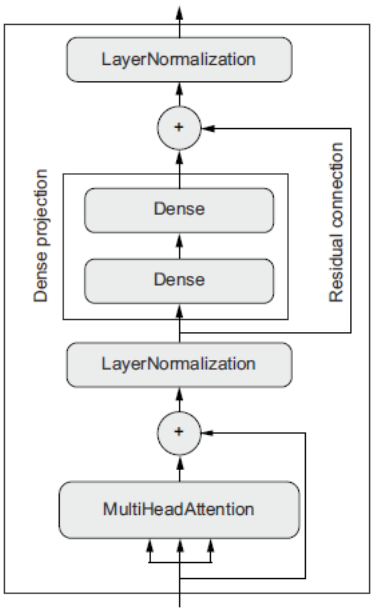


Figure 3.6: Transformer encode schema [3]

As seen in Figure 3.6, the transformer encoder is composed by a Multi-Head Attention layer together with some Dense layers and it adds normalization layers and residual connections.

Multi-Head Attention Mechanism

Multi-Head Attention (MHAT) consists on applying more than one Self-Attention mechanisms at the same time. Then all the outputs are concatenated, a linear transformation is applied and the result is utilized as the output of the MHAT. Figure 3.7 shows the architecture.

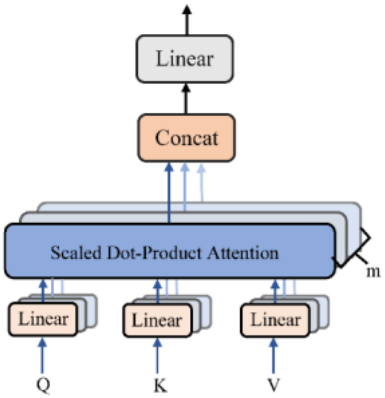


Figure 3.7: Architecture of the MHAT mechanism [4]

As Transformers were originally designed for translation, the Self-Attention mechanism is slightly different to the one described above. In translation, two different sequences should be considered: the one being translated and the target sequence it is being converted to. In order to deal with this problem, the query-key-value model is introduced. Query (Q), key (K) and value (V) are three sentences:

- Query: what the algorithm is looking for
- Value: all the information available in the model
- Key: representation of value that can be compared to the query

so the algorithm match queries and keys, computes how related they are and it returns a weighted sum of the values, obtaining the following formula:

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

where d_k is the dimension of Q and K and the *softmax* function normalizes the obtained weights.

Contrary to the case of translation, in the case of sentiment analysis just one sentence is considered such that each sentence is being compared to itself in order to obtain context information of each token. This one sentence available in sentiment analysis will be at the same time considered as query, key and value so K=V=Q.

Dense and Normalization Layers

After the Multi-Head Attention, a Layer Normalization is applied to the set formed by the inputs and outputs of the MHA. This process normalize the matrix representation of each sequence independently from the other sequences, helping to accelerate and stabilize the learning process. After this step there are two Dense or Fully Connected Layers and later another Layer Normalization is applied as shown in Figure 3.6.

Chapter 4

Model Applications and Results

In this chapter, five different models are evaluated using a Twitter data set with the objective of analyzing the relevance of attention mechanism on sentiment analysis. In the first section the dataset is presented and later the models are described. Finally, the results for each model are considered and compared.

4.1 Data set

For the last two years, there are an average of 867 million of new tweets per day [11] what makes Twitter one of the main source of opinionated short texts. However, in order to create an original dataset for sentiment analysis first classification has to be done manually so it is very difficult to create new data sets with enough data to train a neural network. Hence we will be working with Sentiment140[9], a publicly available data set.

Tweet texts often contain user mentions, hyperlinks, non-letter characters and punctuation. Before use the texts in the neural networks, some cleaning should be done. This does not change the sentiment of the sentence and it makes the embedding easier. It includes

- Convert text to lower case
- Remove symbols of line breaks (\n and \r)
- Remove sings of retweets (re:)

- Remove the non utf-8 characters
- Remove mentions, hashtags and hyperlinks
- Remove all punctuation symbols
- Remove numbers
- Remove multiple spaces

	sentiment	text	clean text
1599773	1	@ijustine hey	hey
1599776	1	@linksforluv you betcha!!	you betcha
1599803	1	@SanctumInc right-click, Repost.	rightclick repost
1599817	1	@YouLuvMe sure..... bighead	sure bighead
1599907	1	@gabespears morning	morning
1599914	1	@PJA4ever Back..	back
1599930	1	@AndrewDearling *yawns*	yawns
1599963	1	@OHTristaN it's sunoudy	its sunoudy
1599993	1	@SCOOBY_GRITBOYS	

Figure 4.1: Sentences with less than three words

After cleaning, duplicated tweets are deleted and we keep in the data set just the tweets containing three or more words as most of tweets with less than three words are not full sentences as can bee seen on Figure 4.1. Therefore, the data set is composed by a total of 1.526.942 distributed in the following way:

Sentiment	Training Set	Test Set	Total number of Tweets
Negative	693.831	77.093	770.924
Positive	680.416	75.602	756.018

Twitter data is very unique due to the informal language being used but also because of its length as each tweet has a limit of 140 characters[1]. Figure 4.2 shows the distribution of tweets according to their number of words. Tweets are very similar among sentiments regarding length and, in both cases, most of the tweets are between 5 and 20 words.

[1]From 2017 this limit is 280 [1] but the data used in this thesis was extracted in 2009

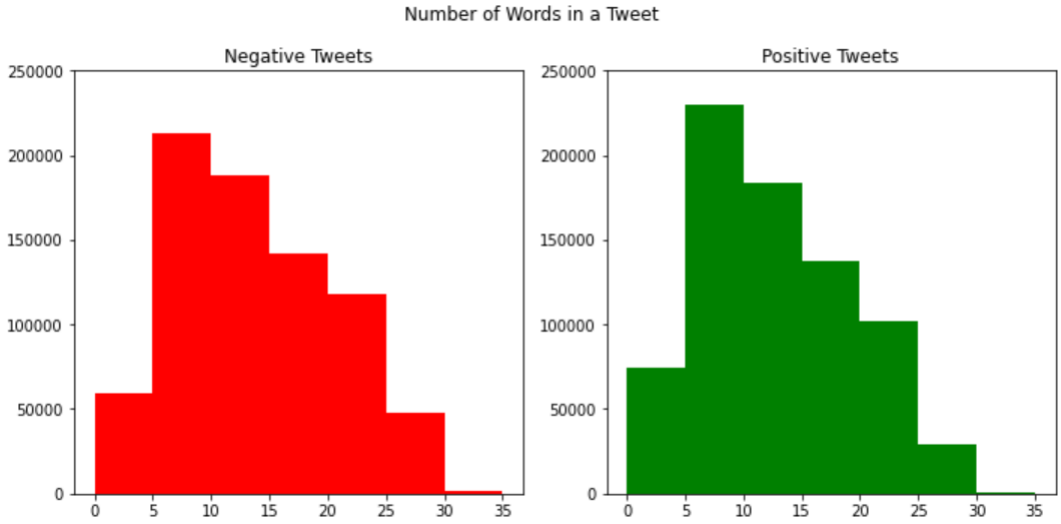


Figure 4.2: Number of words in a Tweet

Figure 4.3 shows the most common words in tweets for each sentiment without considering stopwords (a, all, also, am, else, had, not...). Words associated with positive feelings like *thank*, *love*, *fun*, *nice* are among the most common words for positive tweets while others related with negative feelings like *miss*, *sad* or *hate* are more common on negative tweets.

Figure 4.3: Most common words for each sentiment

Before inputting the data into the neural network, all the the tweets are split in words and a padding function is applied in order to being able to convert them into a matrix by adding 0's at the beginning of the sentence until reaching the equivalent to 35 words. Then, GloVe embedding is applied.

In this case, we are using a pretrained GloVe embedding that has been trained

using Twitter data to have a better representation of the informal language used on social media. For instance it includes words like *bday* or *xoxo*. This embedding covers 70.09% of the words that are used in the considered dataset. Some of the words appearing on the data set not covered by the pretrained GloVe embedding are shown in Figure 4.4. They are words with repeated characters or misspelled.

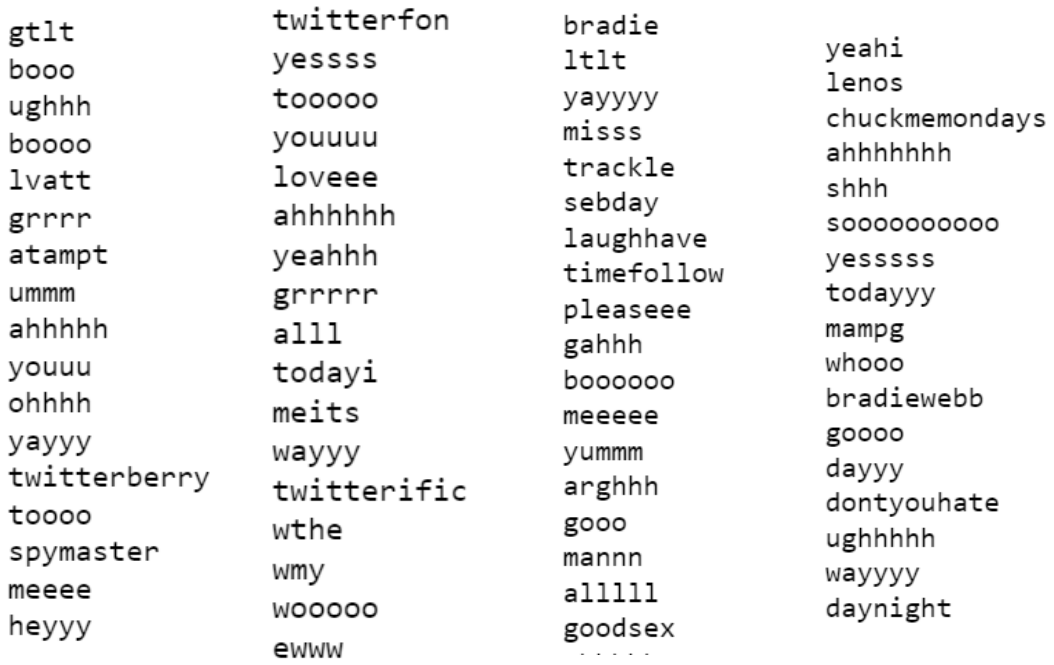


Figure 4.4: Example of words not appearing on GloVe database

The performance of the different models will be evaluated by using a test data set so the original data is split into train and test. Test dataset is selected randomly by maintaining the proportion of tweets for each sentiment and it contains 10% of the total data.

4.2 Models

During the training process, a validation data set is randomly chosen and it represents a 10% of the training data set. In order achieve the best results of the experiment, some parameters of the different models are selected by using grid search and cross-validation. Additionally, to avoid over-fitting, an early stopping mechanisms is used based on the loss of the validation data with a patient of 5 epochs.

4.2.1 CNN Model

The first model to be considered is a CNN with the structure shown on Figure 4.5. The number of filters and kernel size were defined by using a greed search procedure. However model's accuracy is not highly influenced by the number of filters or kernel size obtaining a difference between the best and worst model of less than 1% on the validation data set.

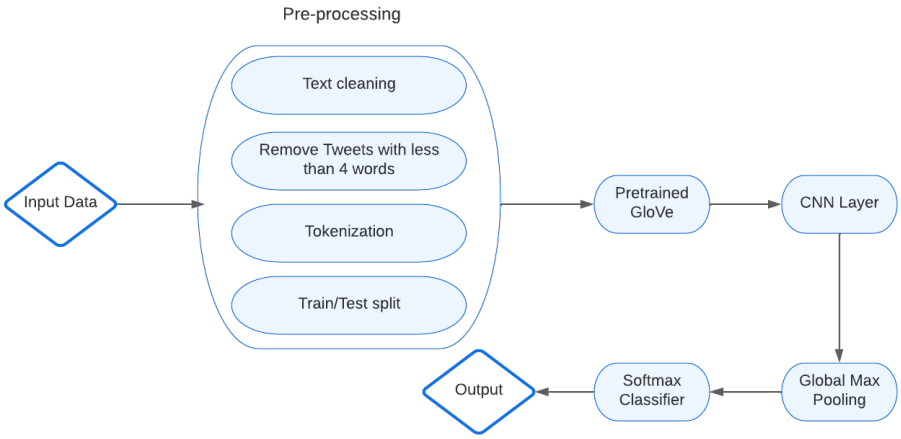


Figure 4.5: Structure of CNN model

Hyperparameter	Value
Embedding size	200
Number of words per tweet	35
Number of filters	64
Kernel size	5
Convolutional layer activation function	ReLU
Padding	same
Kernel regularizer	L1
Optimizer	Adam
Loss function	Binary cross-entropy

Table 4.1: Hyperparameters for CNN Model

The model has been trained using the hyperparameters described on Table 4.1 and Figure 4.6 shows the details of the model where GloVe embedding represents

the first layer where the each tweet is converted into a vector of 200 elements for each word and the sentence maximum length is 35. Therefore, it results a 35 x 200 matrix. The convolutional layer defines 64 filters of kernel size equals to 5, allowing to train 64 different features. Its activation function is *ReLU* and it is considering 'same' padding. It also uses a L1 kernel regularizer with a weight of 0.001 to reduce overfitting. After the CNN layer, the global maximum layer is used to reduce the complexity of the output and prevent overfitting of the training data. The output matrix has a size of 1 x 64. Lastly, a dense layer with *softmax* activation is applied and it outputs a vector with two real numbers between 0 and 1 that represents the probability of the tweet of being negative and positive respectively.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding (Embedding)        (None, 35, 200)           78682200

 conv1d_1 (Conv1D)            (None, 35, 64)            64064

 global_max_pooling1d_1 (Glo  (None, 64)                0
 balMaxPooling1D)

 dense_1 (Dense)              (None, 2)                 130

=================================================================
Total params: 78,746,394
Trainable params: 64,194
Non-trainable params: 78,682,200
_________________________________________________________________
```

Figure 4.6: Details of the CNN model

The model has a total of 64,194 trainable parameters and it needed to be trained for a total of 85 epochs using a batch of 1,000 tweets to obtaining an accuracy of 79.88% measured on the test data set. Figure 4.7 shows the confusion matrix normalized for true values where it can be seen that it performs better on classifying tweets with negative sentiment.

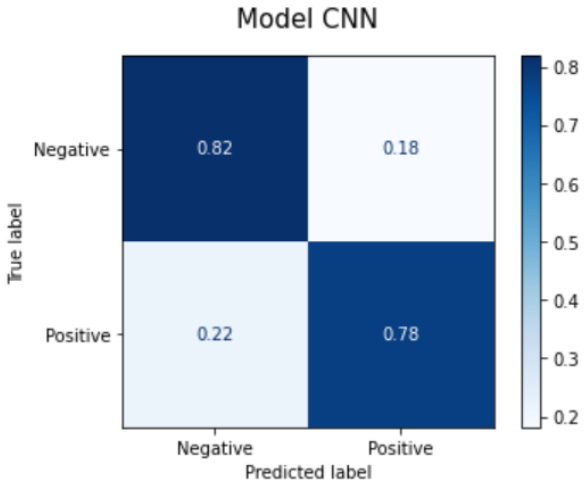


Figure 4.7: Confusion matrix for CNN model

4.2.2 BiLSTM Model

The second model is a BiLSTM model with the structure shown on Figure 4.8. The number of units of the BiLSTM layer has been decided using grid search and cross-validation in order to find the model with better performance. Similar to the case of CNN model, the accuracy measured on the validation set is quite stable regarding the number of filters and the difference of accuracy between the best and worst model is lower than 1%.

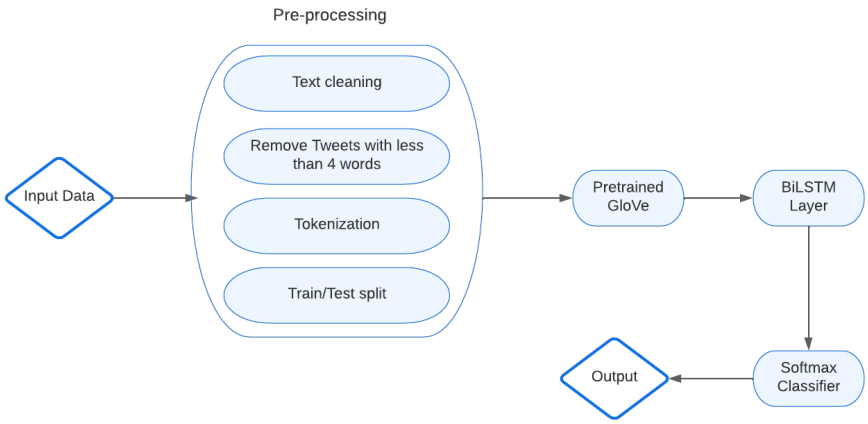


Figure 4.8: Structure of BiLSTM model

Figure 4.9 shows the summary of the model. After the embedding layer a BiLSTM layer is applied with 64 units. There is a dropout of 0.2 inside the BiLSTM layer and another one of 0.5 after it in order to prevent overfitting. The output of the BiLSTM is the input of the dense layer.

Hyperparameter	Value
Embedding size	200
Number of words per tweet	35
BiLSTM units	64
BiLSTM dropout	0.2
Dropout	0.5
Optimizer	Adam
Loss function	Binary cross-entropy

Table 4.2: Hyperparameters for BiLSTM Model

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding (Embedding)        (None, 35, 200)           78682200

 bidirectional (Bidirectiona  (None, 128)               135680
 l)

 dropout (Dropout)            (None, 128)               0

 dense_2 (Dense)              (None, 2)                 258

=================================================================
Total params: 78,818,138
Trainable params: 135,938
Non-trainable params: 78,682,200
_________________________________________________________________
```

Figure 4.9: Details of the BiLSTM model

The model has a total of 135,938 trainable parameters and it needed to be trained for a total of 26 epochs to obtaining an accuracy of 83.54% measured on the test data set. Figure 4.10 shows the confusion matrix normalized for true values.

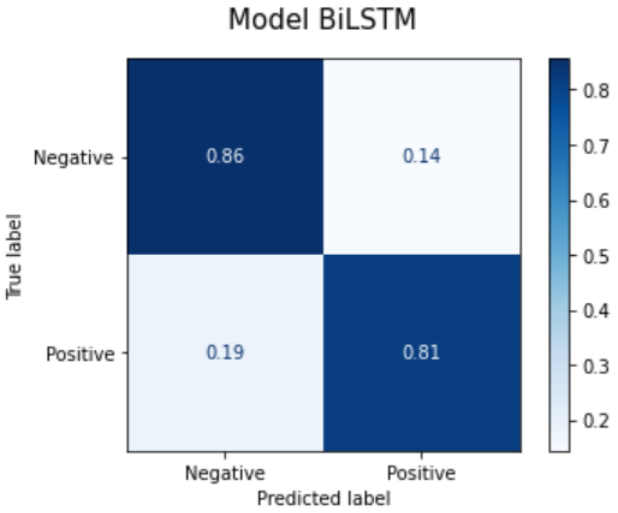


Figure 4.10: Confusion matrix for BiLSTM model

4.2.3 Model combining CNN and BiLSTM

After evaluating CNN and BiLSTM in separate models, this model combines both of them so that, after capturing the features extracted using CNN, a BiLSTM layer is applied in order to filter the information. The number of filters, kernel size and units in BiLSTM have been selected by a grid search procedure with cross-validation and, even if the difference of accuracy among models depending on the hyperparameters is bigger than in the previous cases, it is under 1.5%.

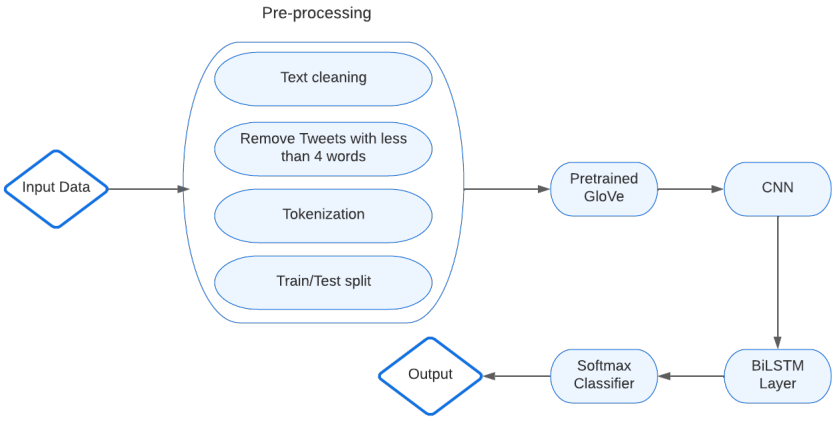


Figure 4.11 shows the summary of the model. In this case, there is a convolutional layer that takes as input the matrix built by the embedding layer and 64 filters of size 3 are applied outputting a 35 x 64 matrix. Then a BiLSTM layer with 16 units is applied to filter the information using its three gates (forget, input and output). In order to avoid overfitting a dropout of 0.2 is set in the BiLSTM layer and, after the BiLSTM layer, there is another dropout of 0.5. The output of the BiLSTM is the input of the dense layer with dimension 2 and *softmax* as activation function.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding (Embedding)        (None, 35, 200)           78682200

 conv1d_2 (Conv1D)            (None, 35, 64)            38464

 bidirectional_1 (Bidirectio  (None, 32)                10368
 nal)

 dropout_1 (Dropout)          (None, 32)                0

 dense_3 (Dense)              (None, 2)                 66

=================================================================
Total params: 78,731,098
Trainable params: 48,898
Non-trainable params: 78,682,200
_________________________________________________________________
```

Figure 4.11: Details of the CNN-BiLSTM model

Hyperparameter	Value
Embedding size	200
Number of words per tweet	35
Number of filters	64
Kernel size	3
Convolutional layer activation function	ReLU
Padding	same
Kernel regularizer	L1
BiLSTM units	16
BiLSTM dropout	0.2
Dropout	0.4
Optimizer	Adam
Loss function	Binary cross-entropy

Table 4.3: Hyperparameters for CNN-BiLSTM Model

As shown in the summary there are a total of 48,898 trainable parameters and it needed to be trained for 22 epochs obtaining an accuracy measured in the test set of 79.87%. Figure 4.12 shows the confusion matrix of the model normalized for true values.

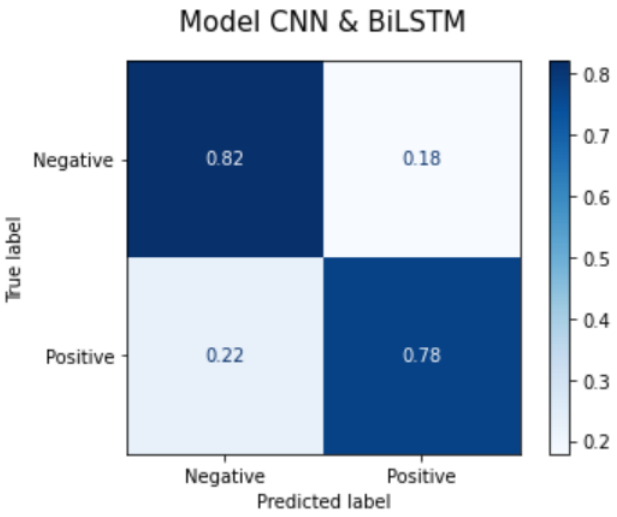


Figure 4.12: Confusion matrix for CNN-BiLSTM model

4.2.4 Model BiLSTM with Self-attention

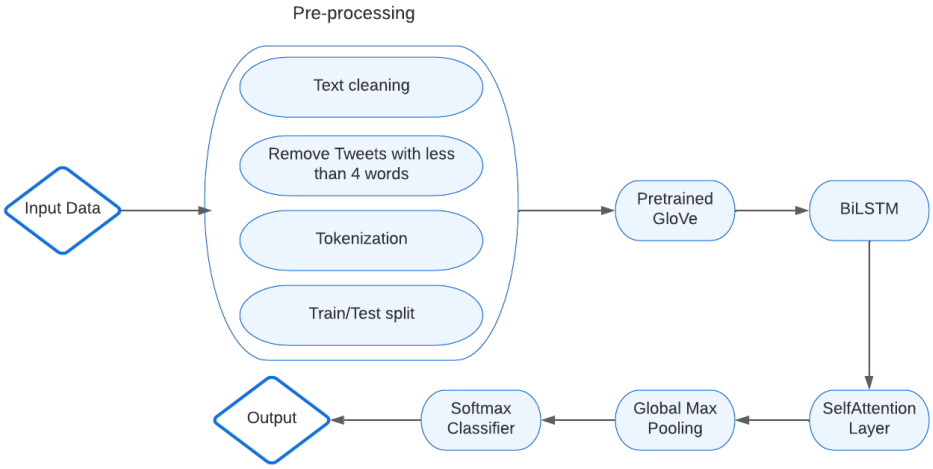


Figure 4.13: Structure of BiLSTM with Self Attention model

The model described in Figure 4.13 uses Self Attention after the BiLSTM layer for which the number of units has been chosen using grid search with cross-validation. The difference between models changing the number of units in the BiLSTM is up to 2% of accuracy (measured on the validation set).

Figure 4.14 shows the summary of the model and Table 4.4 the details of the hyper-parameters. In this model, Self-Attention is applied to the hidden states produced by the BiLSTM layer obtaining a new representation of the input sentence. After the attention application, a global maximum layer and dropout are used to reduce the complexity of the output and prevent overfitting.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding (Embedding)        (None, 35, 200)           78682200

 bidirectional_5 (Bidirectio  (None, 35, 128)          135680
 nal)

 seq_self_attention_3 (SeqSe  (None, 35, 128)          8257
 lfAttention)

 global_max_pooling1d_5 (Glo  (None, 128)              0
 balMaxPooling1D)

 dropout_5 (Dropout)          (None, 128)              0

 dense_7 (Dense)              (None, 2)                258

=================================================================
Total params: 78,826,395
Trainable params: 144,195
Non-trainable params: 78,682,200
_________________________________________________________________
```

Figure 4.14: Details of the BiLSTM with Self Attention model

Hyperparameter	Value
Embedding size	200
Number of words per tweet	35
BiLSTM units	64
BiLSTM dropout	0.3
Dropout	0.4
Optimizer	Adam
Loss function	Binary cross-entropy

Table 4.4: Hyperparameters for BiLSTM with Self Attention model

In this case the number of trainable parameters is 144,195 and it has been trained for 29 epochs before the early stopping method interrupted the process. When evaluating this model on the test set, the accuracy obtained is 83,74% and, similarly to the previous models, negative tweets are classified better than positive ones as shown on Figure 4.15. However, compared with the model that applies just BiLSTM, without attention, the accuracy on negative tweets is the same but the one for positive tweets is higher now.

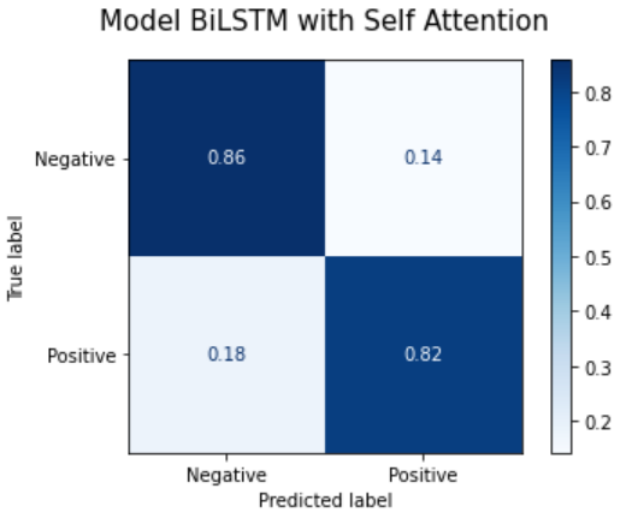
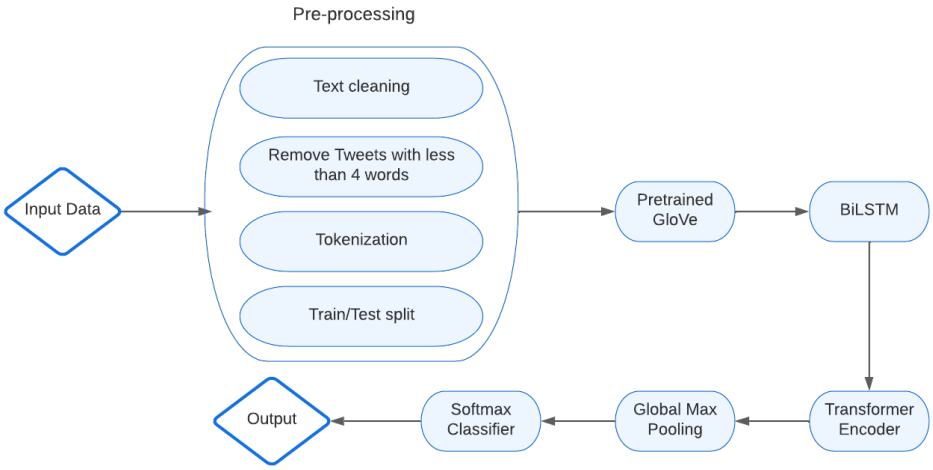


Figure 4.15: Confusion matrix for BiLSTM with Self Attention model

4.2.5 Model BiLSTM with Transformer Encoder

The last model combines BiLSTM with a Transformer Encoder. This model is more complex than the previous ones as can be seen in the number of trainable parameters that are 576,434 for this model. The number of heads of the transformer encoder as well as the dimension of the first dense unit inside the transformer are defined by using grid search and cross-validation. In this case the difference between the best and worst performing models is of almost 8% in terms of accuracy measured in the validation set. The hyperparameters chosen to train the model are shown on Table 4.5.



Hyperparameter	Value
Embedding size	200
Number of words per tweet	35
BiLSTM units	100
BiLSTM dropout	0.3
Transformer number of heads	2
Transformer dense dimensions	32 and 100
Multi-Head Attention regularization	L1
Dropout	0.4
Optimizer	Adam
Loss function	Binary cross-entropy

Table 4.5: Hyperparameters for BiLSTM with Transformer Encoder model

Figure 4.16 shows the details to the model. In this case, the matrix generated by the embedding layer is fed into an BiLSTM layer with 100 units. Then the outputs are taken as the input of the transformer encoder that applies first MHAT with 2 heads, then normalization is applied to the inputs and outputs of MHAT together. Next step two Dense layers are applied, the first one with dimension 32 and the second one with 200 in order to match the dimensions of the input. The output is normalized and passed to the next layer as the output of the transformer encoder layer. Later a Global Max Pooling and the last Dense layer for classification are applied.

As it is a complex model with a high number of trainable parameters, it is easy

that it overfits the training set. In order to avoid that, several dropouts were included during the whole process together with the regularization layers and L1 regularization inside the transformer encoder.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 35, 200)           78682200

 bidirectional_99 (Bidirecti  (None, 35, 200)          240800
 onal)

 dropout_382 (Dropout)       (None, 35, 200)           0

 transformer_block_97 (Trans  (None, 35, 200)          335232
 formerBlock)

 dropout_385 (Dropout)       (None, 35, 200)           0

 global_max_pooling1d_95 (Gl  (None, 200)              0
 obalMaxPooling1D)

 dense_296 (Dense)           (None, 2)                 402

=================================================================
Total params: 79,258,634
Trainable params: 576,434
Non-trainable params: 78,682,200
_________________________________________________________________
```

Figure 4.16: Details of the BiLSTM with Transformer Encoder model

Figure 4.17 shows the confusion matrix for the model. In this case, unlikely the previous ones, the performance classifying positive and negative tweets is equal.

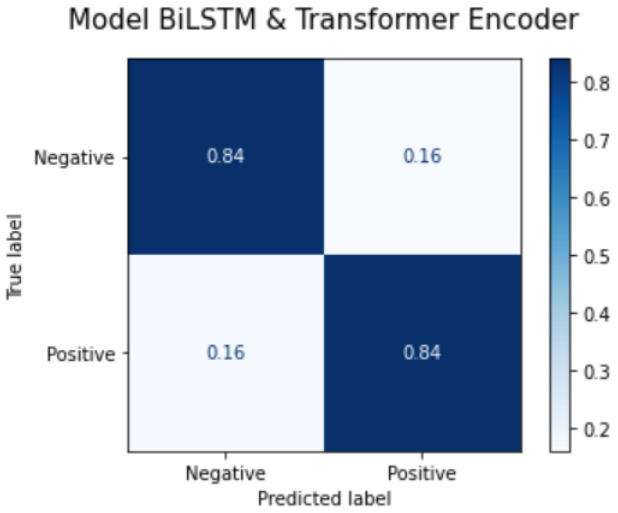


Figure 4.17: Confusion matrix for BiLSTM with Transformer Encoder model

4.3 Results comparison

In order to evaluate the models and compare them, a test data set containing more than 153,000 tweets is used. The evaluation standards are Precision (Pr), Recall (Re), F1 measure and Accuracy (Acc) defined as follows:

- $Pr = \frac{TP}{TP+FP}$
- $Re = \frac{TP}{TP+FN}$
- $F1 = 2 \times \frac{Pr \times Re}{Pr+Re}$
- $Acc = \frac{\text{Correct classifications}}{\text{Total Classifications}}$

Where TP, TN, FP and FN stand for true positive, true negative, false positive and false negative respectively. Table 4.6 shows a summary of all these metrics for each one of the models considered for comparison.

Model	Class	Precision	Recall	F1	Accuracy
CNN	Negative	0.79	0.82	0.80	0.7988
	Positive	0.81	0.78	0.79	
BiLSTM	Negative	0.82	0.86	0.84	0.8354
	Positive	0.85	0.81	0.83	
CNN + BiLSTM	Negative	0.79	0.82	0.80	0.7987
	Positive	0.81	0.78	0.79	
BiLSTM with Self Attention	Negative	0.83	0.86	0.84	0.8374
	Positive	0.85	0.82	0.83	
BiLSTM + Transformer Encoder	Negative	0.84	0.84	0.84	0.8379
	Positive	0.84	0.84	0.84	

Table 4.6: Results obtained for the different models

Model	Number of trainable parameters
CNN	64,194
BiLSTM	135,938
CNN-BiLSTM	48,898
BiLSTM with Self Attention	144,195
BiLSTM + Transformer Encoder	576,434

Table 4.7: Number of trainable parameters for each model

In addition to the evaluation metrics, the number of trainable parameters should be taken into account when comparing two different models. By doing this, we can better choose between two models with similar performance but different number of trainable parameters. This is the case of the models CNN and CNN-BiLSTM where the difference in terms of accuracy is almost insignificant but, the first one has almost 20,000 more trainable parameters as shown in Table 4.7. At the same time, CNN needs to be trained for 85 epochs while CNN-BiLSTM needs just 22 so we can say that applying a BiLSTM after a CNN is a better option than train a more complex CNN for this dataset.

However, based on the results we can see that BiLSTM works better by itself that after a CNN layer. The difference on the accuracy of these two models is slightly higher than 3.5%.

Differences in performance of BiLSTM, BiLSTM with Self Attention and BiLSTM with Transformer encoder is of less than 0.25% while the difference in the number of parameters is huge (BiLSTM with Transformer Encoder has around four times the number of trainable parameters of the other two models). Even if the performance of these three models is really similar, we can see differences when observing the

confusion matrices. If we consider the results for the models that apply attention after the BiLSTM layer (Figures 4.15 and 4.17) we can see that the one containing Transformer Encoder performs better than the one with Self Attention for negative tweets but worst on positive ones.

Figure 4.18 shows the accuracy evolution during the training process in both the test and validation sets for all the models considered. We can see the difference of epochs needed for each model to arrive to its better performance. BiLSTM with Transformer Encoder (Transformer for short in the graph) is the one with a higher performance and the one requiring less epochs to reach it. However, due to its complexity, each one of this epochs require more computational time than any of the other models.

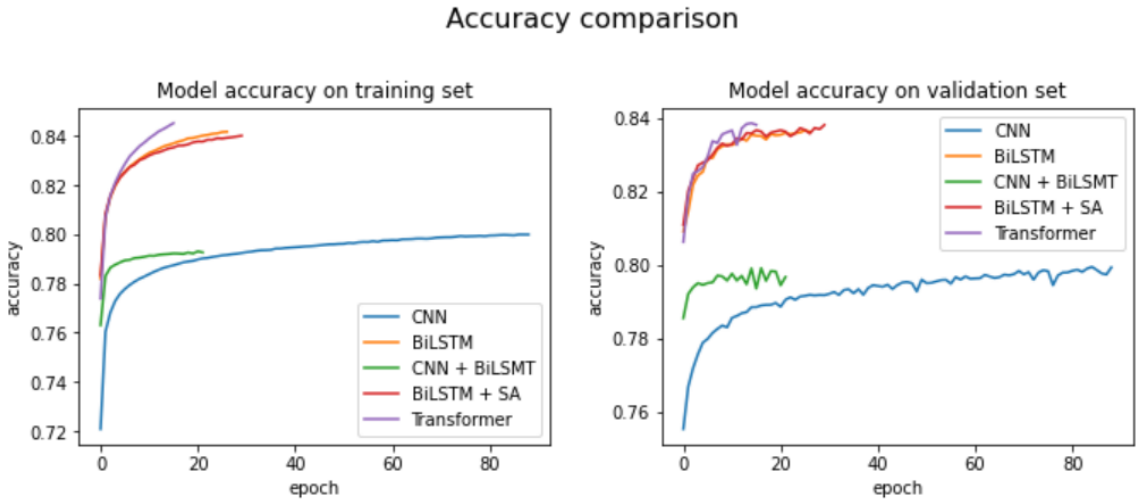


Figure 4.18: Accuracy evolution

Taking the better performing model's confusion matrix (BiLSTM with Transformer Encoder), word clouds of each cell (TP, TN, FP and FN) was done and it is shown on Figure 4.19. By observing it we can not see big differences in the most common words for each cell, apart from informal abbreviations of words like *lol*, *im* or *u* being more common on misclassified tweets. However the word clouds just present a selection of the most common words but rare words like the ones misspelled like the ones not covered by the pretrained embedding (Figure 4.4) may appear more on misclassified tweets.

Figure 4.19: Word cloud for each cell of the confusion matrix for BiLSTM-Transformer Encoder model

Figure 4.20 shows the text of some misclassified tweets together with its real and predicted values. We can see that some are ambiguous and the sentiment is not clear like, for instance, the tweet "@terarenee is what something i did ?". This tweet is labeled as positive but, without context, it can not be associated to a sentiment. Additionally, we have the case of tweets that can be seen as wrongly labeled like "Off I Go! Twitter Is Too Slow Tonight! I'll Be Back On 2mmorow!" where the user is complaining about the speed of Twitter so it can be perceived as something negative even if it is labeled as positive. We can also observe on this tweets informal abbreviations like commented above (*2mmorow*, *u*, *polisci* or *ur*)

Tweet	Real Sentiment	Predicted Sentiment
@terarenee is what something i did ?	Positive	Negative
I'm usually right all the time..... And when I'm wrong, I'm right for being wrong!	Positive	Negative
@_CorruptedAngel my god, really! Really? 50? Christ	Negative	Positive
Off I Go! Twitter Is Too Slow Tonight! I'll Be Back On 2mmorow!	Positive	Negative
@bobmcwhirter I've had 11% and 18%+ beer just last week, and it was indeed evil!	Positive	Negative
@mstiffanyu I love how u put a smiley face after studying for polisci. Good luck! Ur on summer already	Negative	Positive
I will try. She doesn't seem in the mood to go though. We'll see	Positive	Negative

Figure 4.20: Tweets misclasified for BiLSTM-Transformer Encoder model

Chapter 5

Conclusions and Future Work

In this master thesis the objective was to study the impact of attention mechanisms in deep learning models for sentiment analysis. In order to do that, 5 models were presented, two of them containing attention mechanisms, and they were used in sentiment classification for tweets. Even if the final study has been done considering just positive and negative sentiment, we start trying the models also considering neutral tweets. The problem was that neutral tweets came form a different data set and they all belong to very specific topics so the algorithms have a precision of 99% on neutral tweets as they learn to identify the topic while the precision for positive and negative tweets were lower than 80% so we decided discard neutral tweets.

In Chapter 4 we have seen that when comparing the models containing BiLSTM (with or without attention) with the CNN model there is an improve on the accuracy as well as on the number of epochs needed for the model to train. However, the differences in terms of accuracy between the model just containing BiLSTM and the ones introducing attention mechanisms are almost insignificant while the computational cost increase for the last ones.

On conclusion, when using a Twitter data set where all the sentences are under 140 characters and almost 80% of the tweets contain 20 words or less, the memory of BiLSTM could be enough to capture all the dependencies between words in the sentences as there is not a big difference in the performance of model with just a BiLSTM layer and the ones that added attention mechanisms on top while the computational cost do significantly increase.

Next step on this study will be to complete the pretrained GloVe embedding with the missing words so that all the words in the data set can have a representation in the embedding-space. Other approach can be to visualize the weights inside the attention mechanisms in order to understand to which parts of the sentences they are paying more attention and compare it with BiLSTM.

Bibliography

[1] K. Dijkstra et al. A.B Boot E. Tjong Kim Sang. "How character limit affects language usage in tweets". In: *Palgrave Commun* 5 (2019).

[2] A. S. M. Alharbi and E. de Doncker. "Twitter sentiment analysis with a deep neural network: An enhanced approach using user behavioral information". In: *Cognitive Systems Research* (2019).

[3] F. Chollet. *Deep Learning with Python*. Manning Publications, (2017).

[4] H. S. Sharaf Al-deen and Z. Zeng et al. "An Improved Model for Analyzing Textual Sentiment Based on a Deep Neural Network Using Multi-Head Attention Mechanism". In: *Applied System Innovation* (2021).

[5] *Differences between Bidirectional and Unidirectional LSTM*. URL: `https://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm` (visited on 12/11/2022).

[6] *Evaluate the Performance of Deep Learning Models in Keras*. URL: `https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/` (visited on 09/01/2022).

[7] D. Foster. *Generative Deep Learning: Teaching Machines to Paint, Write, Compose and Play*. O'Reil, (2019).

[8] *GloVe: Global Vectors for Word Representation*. URL: `https://nlp.stanford.edu/projects/glove/` (visited on 11/06/2022).

[9] A. Go, R. Bhayani, and L. Huang. "Twitter sentiment classification using distant supervision". In: *CS224N project report, Stanford* 1.12 (2009).

[10] A. Hassan and A. Mahmood. "Deep Learning approach for sentiment analysis of short texts". In: *3rd International Conference on Control, Automation and Robotics (ICCAR)* (2017).

[11] *How Many Tweets per Day 2022 (New Data)*. URL: `https://www.renolon.com/number-of-tweets-per-day/` (visited on 12/26/2022).

[12] *Illustrated: Self-Attention*. URL: `https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a` (visited on 09/01/2022).

[13] J. Pennington, R. Socher, and C. Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP).* (2014).

[14] J. Qian, Z. Niu, and C. Shi. "Sentiment Analysis Model on Weather Related Tweets with Deep Neural Network". In: *Association for Computing Machinery* (2018).

[15] A. Vaswani and N. Shazeer et al. "Attention is All You Need". In: *Curran Associates Inc.* (2017).

[16] R. Wadawadagi and V. Pagi. "Sentiment analysis with deep neural networks: comparative study and performance assessment". In: *Springer Nature B.V. 2020* (2020).

[17] J. Xie and B. Chen et al. In: *Self-Attention-Based BiLSTM Model for Short Text Fine-Grained Sentiment Classification* 7 (2019).

[18] L. Zhang, S. Wang, and B. Liu. "Deep Learning for Sentiment Analysis : A Survey". In: *CoRR* abs/1801.07883 (2018).

Python Code

```
##############################
######## Packages ##########
##############################

import numpy as np
import pandas as pd
import os
import operator
import nltk
import joblib
import tensorflow as tf
import sklearn.model_selection
import re
import string
import plotly.express as px
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline
pd.options.plotting.backend = "plotly"

from tensorflow.keras import layers
from keras.preprocessing.text import Tokenizer

from sklearn.model_selection import train_test_split, GridSearchCV
    , StratifiedKFold
from keras.models import Sequential, Model
from keras_preprocessing.sequence import pad_sequences
from keras.layers import Dense, Embedding,  Conv1D, Bidirectional,
     Dropout, LSTM, GlobalMaxPooling1D, MultiHeadAttention,
    LayerNormalization, Layer
from keras.regularizers import L1
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.models import load_model
from sklearn.metrics import confusion_matrix, f1_score,
    classification_report, ConfusionMatrixDisplay
from keras_self_attention import SeqSelfAttention
```

```
from scikeras.wrappers import KerasClassifier
from tensorflow.keras.utils import to_categorical
from wordcloud import WordCloud, STOPWORDS

os.chdir('/home/USERS/elena.blanco.gonzalez/TFM/Data')

############################
###### Loading Data #######
############################

colnames = ['sentiment', 'ID', 'data', 'flag','user','text']
df = pd.read_csv('training1600000.csv', names = colnames, header =
    None, encoding = 'latin-1')
df = df[['sentiment','text']]
df['sentiment'] = df['sentiment'].map({0:0, 4:1})
df.shape

df.drop_duplicates(subset = 'text',inplace = True) #Eliminate
   duplicates if any
df = df.loc[df['sentiment'].isnull() == False] #Keep just labeled
   tweets
df.shape , df.head()

############################
###### Data Cleaning #######
############################

def clean_text(text):
  text = str(text)
  text = text.lower()
  text = re.sub('\r|\n', ' ',text) #Remove \n and \r
  text = re.sub('re:','',text) #Replace signs of RTs
  text = re.sub(r'[^\x00-\x7f]',r' ', text) #remove non utf8
  text = re.sub(r'&quot',r' ', text) #remove non utf8
  text = re.sub(r'http\S+|www\S+|https\S+', ' ', text, flags=re.
   MULTILINE) #remove URLs
  text = re.sub(r'\@\w+|\#',' ', text) #Remove hashtags
  text = ''.join([i for i in text if i not in string.punctuation])
  text = re.sub(r'1|2|3|4|5|6|7|8|9|0', ' ',text) #Remove numbers
  text = re.sub('\s\s+' ,' ',  text) #remove multiple spaces
  return text

df['clean text'] = df['text'].apply(clean_text)
df.head()

df['clean text'].replace('', np.nan, inplace=True)
df.dropna(subset = ['clean text'], inplace=True)

df_short = df[df['clean text'].str.split().str.len() <3] #Getting
```

```
    rid of sentences with less than 3 words
df = df[df['clean text'].str.split().str.len() > 2] #Getting rid
    of sentences with less than 3 words
df.shape
df_short.tail(10)

##################################
### Exploratory Data Analysis ###
##################################

def plot_hist_classes(df, header):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (10,5))
    df_split = df[df['sentiment'] == 0]['clean text'].str.split()
    df_len = df_split.apply(lambda x: len(x))
    ax1.hist(df_len,color = 'red', range = [0, 35],bins = np.
    arange(0, 40, 5))
    ax1.set_ylim([0, 250000])
    ax1.set_title('Negative Tweets')
    df_split = df[df['sentiment'] == 1]['clean text'].str.split()
    df_len = df_split.apply(lambda x: len(x))
    ax2.hist(df_len,color = 'green', range = [0, 35], bins = np.
    arange(0, 40, 5))
    ax2.set_ylim([0, 250000])
    ax2.set_title('Positive Tweets')
    fig.suptitle(header)
    fig.tight_layout()
    plt.show()
    plt.close()

plot_hist_classes(df, header='Number of Words in a Tweet')

df_negative = df[df['sentiment'] == 0]['clean text'].tolist()
df_positive = df[df['sentiment'] == 1]['clean text'].tolist()
negative = ' '.join(df_negative)
positive = ' '.join(df_positive)

stopwords = set(STOPWORDS)
def plot_WordClouds(df_neg, df_pos):
    wc = WordCloud(background_color = "white", max_words = 50,
    width = 900, height = 500, stopwords = stopwords,colormap="
    autumn")
    fig,(ax1,ax2) = plt.subplots(1, 2, figsize = (15,8))
    wc.generate(df_neg)
    ax1.imshow(wc)
    ax1.axis("off")
    ax1.set_title('Negative tweets')
    wc = WordCloud(background_color = "white", max_words = 50,
    width = 900, height = 500, stopwords = stopwords,colormap="
    summer")
```

```
    wc.generate(df_pos)
    ax2.imshow(wc, interpolation='bilinear')
    ax2.axis("off")
    ax2.set_title('Positive tweets')
    plt.show()
    plt.close()

plot_WordClouds(negative, positive)
df['sentiment'].value_counts()

##################################
####### Train/Test split ########
##################################

Train_text,Test_text , Train_sentiment,  Test_sentiment=
    train_test_split(df['clean text'],df['sentiment'], random_state
    =123,stratify=df['sentiment'], train_size = .9)
Train_text.shape, Test_text.shape

Train_text.head()
Train_sentiment.value_counts()
Test_sentiment.value_counts()

##################################
######### Tokenization ##########
##################################

tokenizer = Tokenizer()
tokenizer.fit_on_texts(df['clean text'])

word_index = tokenizer.word_index
vocab_size = len(word_index) + 1
vocab_size

max_seq_len = 35
X_train = pad_sequences(tokenizer.texts_to_sequences(Train_text),
    maxlen = max_seq_len)
X_test = pad_sequences(tokenizer.texts_to_sequences(Test_text),
    maxlen = max_seq_len)
X_train.shape , X_test.shape

Y_train = to_categorical(Train_sentiment, num_classes=2)
Y_test = to_categorical(Test_sentiment,  num_classes=2)
Y_train.shape, Y_test.shape

##################################
########## Embedding ############
##################################

```

```
glove_tw = 'glove.twitter.27B.200d.txt'
embed_dim = 200
embeddings_index = {}

f = open(glove_tw)
for line in f:
  values = line.split()
  word =  values[0]
  coefs = np.asarray(values[1:], dtype='float32')
  embeddings_index[word] = coefs
f.close()

count = 0
for key in word_index.keys():
    if key not in embeddings_index:
        count += 1
        print(key)

print(np.round(count/vocab_size*100,2), '%') #Percentage of the
    vocabulary covered by the embedding

embedding_matrix = np.zeros((vocab_size, embed_dim))
for word, i in word_index.items():
  embedding_vector = embeddings_index.get(word)
  if embedding_vector is not None:
    embedding_matrix[i] = embedding_vector

embedding_layer = Embedding(vocab_size, embed_dim, weights = [
    embedding_matrix], input_length = max_seq_len, trainable =
    False)

##################################
########## CNN Model #############
##################################

######## Grid Search for hyperparameters

def create_model(filters, kernel_size):
    # Create model
    model = Sequential([
    embedding_layer,
    Conv1D(filters, kernel_size, activation = 'relu',
    kernel_regularizer = L1(l = 0.001), padding = 'same'),
    GlobalMaxPooling1D(),
    Dropout(0.4),
    Dense(2, activation = 'softmax')
    ])
    # Compile model
    model.compile(loss = 'binary_crossentropy', optimizer = 'adam'
```

```
    , metrics = ['accuracy'])
    return model

model = KerasClassifier(model = create_model, epochs = 20,
    batch_size = 1000, verbose = 0)
# Define the grid search parameters
filters = [16,32,64]
kernel_size = [3,5,7]
param_grid = dict(model__filters = filters, model__kernel_size =
    kernel_size)
grid = GridSearchCV(estimator = model, param_grid = param_grid,
    n_jobs = 1, cv = 3)
grid_result = grid.fit(X_train, Y_train, validation_split = 0.1)
# Summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.
    best_params_))
means = grid_result.cv_results_['mean_test_score']
params = grid_result.cv_results_['params']
for mean, param in zip(means, params):
    print("%f with: %r" % (mean, param))

######## Model Definition and Training

model_cnn = Sequential([
    embedding_layer,
    Conv1D(64,5, activation = 'relu', kernel_regularizer = L1(l =
    0.001),padding = 'same'),
    GlobalMaxPooling1D() ,
    Dense(2, activation = 'softmax')
    ])

model_cnn.compile(loss = 'binary_crossentropy', optimizer = tf.
    keras.optimizers.Adam(learning_rate = 0.0001), metrics = ['
    accuracy'])
es = EarlyStopping(monitor = 'val_loss', mode = 'min', verbose =
    2, patience = 5)
mc = ModelCheckpoint('cnn.h5', monitor = 'val_accuracy', mode = '
    max', verbose = 2, save_best_only = True)
history_cnn = model_cnn.fit(X_train, Y_train, batch_size = 1000,
    epochs = 2000, validation_split = 0.1, callbacks = [es, mc])

best_cnn = load_model('cnn.h5')
best_cnn.summary()

######## Model Evaluation

score_cnn = best_cnn.evaluate(X_test,Y_test)

def plot_training_hist(history, title):
```

```
    fig, ax = plt.subplots(1, 2, figsize=(10,4))
    ax[0].plot(history.history['accuracy'])
    ax[0].plot(history.history['val_accuracy'])
    ax[0].set_title('Model Accuracy')
    ax[0].set_xlabel('epoch')
    ax[0].set_ylabel('accuracy')
    ax[0].legend(['train', 'validation'], loc='best')
    ax[1].plot(history.history['loss'])
    ax[1].plot(history.history['val_loss'])
    ax[1].set_title('Model Loss')
    ax[1].set_xlabel('epoch')
    ax[1].set_ylabel('loss')
    ax[1].legend(['train', 'validation'], loc='best')
    fig.suptitle(title, size=15, y=1)

plot_training_hist(history_cnn, title= 'Model CNN')

Y_pred = best_cnn.predict(X_test)
y_pred = np.argmax(Y_pred, axis = 1)
y_test = np.argmax(Y_test, axis = 1)

print('Model CNN')
print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred, normalize = 'true')
labels = ["Negative", "Positive"]
display = ConfusionMatrixDisplay(confusion_matrix = cm,
    display_labels = labels)
display.plot(cmap = plt.cm.Blues)
display.ax_.set_title("Model CNN",  size = 15, y = 1.05)

#################################
######## BiLSTM Model ###########
#################################

######## Grid Search for hyperparameters

def create_model(units):
    # Create model
    model = Sequential([
        embedding_layer,
        Bidirectional(LSTM(units, dropout=0.2)),
        Dropout(0.5) ,
        Dense(2, activation='softmax')
    ])
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam',
    metrics=['accuracy'])
    return model
```

```
294
295 model = KerasClassifier(model=create_model, epochs=30, batch_size
    =1000, verbose=0)
296 # Define the grid search parameters
297 units = [16,32,64,128]
298 param_grid = dict(model__units=units)
299 grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs
    =1, cv=3)
300 grid_result = grid.fit(X_train, Y_train,validation_split=0.1)
301 # Summarize results
302 print("Best: %f using %s" % (grid_result.best_score_, grid_result.
    best_params_))
303 means = grid_result.cv_results_['mean_test_score']
304 params = grid_result.cv_results_['params']
305 for mean, param in zip(means, params):
306     print("%f with: %r" % (mean, param))
307
308 ######## Model Definition and Training
309
310 model_bilstm = Sequential([
311     embedding_layer,
312     Bidirectional(LSTM(64, dropout=0.2)),
313     Dropout(0.5) ,
314     Dense(2, activation='softmax')
315 ])
316 model_bilstm.compile(loss='binary_crossentropy',optimizer='adam',
    metrics=['accuracy'])
317 es = EarlyStopping(monitor='val_loss', mode='min', verbose=2,
    patience=5)
318 mc = ModelCheckpoint('bilstm.h5', monitor='val_accuracy', mode='
    max', verbose=2, save_best_only=True)
319 history_bilstm=model_bilstm.fit(X_train, Y_train, batch_size=1000
    ,epochs=2000, validation_split=0.1,callbacks=[es, mc])
320
321 best_bilstm = load_model('bilstm.h5')
322 best_bilstm.summary()
323
324 ######## Model Evaluation
325
326 score = best_bilstm.evaluate(X_test,Y_test)
327 plot_training_hist(history_bilstm, title='Model BiLSTM')
328
329 Y_pred = best_bilstm.predict(X_test)
330 y_pred = np.argmax(Y_pred, axis = 1)
331 y_test = np.argmax(Y_test, axis = 1)
332 print('Model BiLSTM')
333 print(classification_report(y_test, y_pred))
334
335 cm = confusion_matrix(y_test, y_pred,normalize='true')
```

```
labels = ["Negative", "Positive"]
display = ConfusionMatrixDisplay(confusion_matrix=cm,
    display_labels=labels)
display.plot(cmap = plt.cm.Blues)
display.ax_.set_title("Model BiLSTM",  size=15, y=1.05)

##################################
###### CNN-BiLSTM Model #########
##################################

######## Grid Search for hyperparameters

def create_model(filters, kernel_size, units):
    # create model
    model = Sequential([
        embedding_layer,
        Conv1D(filters, kernel_size, activation='relu',
    kernel_regularizer=L1(l=0.001),padding='same'),
        Bidirectional(LSTM(units, dropout=0.2)),
        Dropout(0.4) ,
        Dense(2, activation='softmax')
    ])
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam',
    metrics=['accuracy'])
    return model

model = KerasClassifier(model=create_model, epochs=30, batch_size
    =1000, verbose=0)
# define the grid search parameters
filters=[16,32,64]
kernel_size=[3,5,7]
units = [16,32,64,128]
param_grid = dict(model__filters=filters, model__kernel_size=
    kernel_size,model__units=units)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs
    =1, cv=3)
grid_result = grid.fit(X_train, Y_train,validation_split=0.1)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.
    best_params_))
means = grid_result.cv_results_['mean_test_score']
params = grid_result.cv_results_['params']
for mean, param in zip(means, params):
    print("%f  with: %r" % (mean, param))

######## Model Definition and Training

model_cnn_bilstm = Sequential([
```

```
  embedding_layer,
  Conv1D(64,3, activation='relu', kernel_regularizer=L1(l=0.001),
   padding='same'),
  Bidirectional(LSTM(16, dropout=0.2)),
  Dropout(0.4) ,
  Dense(2, activation='softmax')
])
model_cnn_bilstm.compile(loss='binary_crossentropy',optimizer='
   adam',metrics=['accuracy'])
es = EarlyStopping(monitor='val_loss', mode='min', verbose=2,
   patience=5)
mc = ModelCheckpoint('cnn_bilstm_.h5', monitor='val_accuracy',
   mode='max', verbose=2, save_best_only=True)
history_cnn_bilstm=model_cnn_bilstm.fit(X_train, Y_train,
   batch_size=1000 ,epochs=2000, validation_split=0.1,callbacks=[
   es, mc])

best_cnn_bilstm = load_model('cnn_bilstm.h5')
model_cnn_bilstm.summary()

######## Model Evaluation

score = best_cnn_bilstm.evaluate(X_test,Y_test)

plot_training_hist(history_cnn_bilstm, title = 'Model CNN & BiLSTM
   ')

Y_pred = best_cnn_bilstm.predict(X_test)
y_pred = np.argmax(Y_pred, axis = 1)
y_test = np.argmax(Y_test, axis = 1)
print('Model CNN BiLSTM')
print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred,normalize ='true')
labels = ["Negative", "Positive"]
display = ConfusionMatrixDisplay(confusion_matrix = cm,
   display_labels = labels)
display.plot(cmap = plt.cm.Blues)
display.ax_.set_title("Model CNN & BiLSTM",  size = 15, y = 1.05)

########################################
### BiLSTM and Self Attention Model ###
########################################

######## Grid Search for hyperparameters

def create_model(units):
    # create model
    model = Sequential([
```

```
419       embedding_layer,
420       Bidirectional(LSTM(units, dropout=0.2,return_sequences=True)
      ),
421       SeqSelfAttention(),
422       GlobalMaxPooling1D(),
423       Dropout(0.4) ,
424       Dense(2, activation='softmax')
425     ])
426     # Compile model
427     model.compile(loss='binary_crossentropy', optimizer='adam',
      metrics=['accuracy'])
428     return model
429
430 model = KerasClassifier(model=create_model, epochs=100, batch_size
      =1000, verbose=0)
431 # define the grid search parameters
432 units = [16,32,64,128]
433 param_grid = dict(model__units=units)
434 grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs
      =1, cv=3)
435 grid_result = grid.fit(X_train, Y_train,validation_split=0.1)
436 # summarize results
437 print("Best: %f using %s" % (grid_result.best_score_, grid_result.
      best_params_))
438 means = grid_result.cv_results_['mean_test_score']
439 params = grid_result.cv_results_['params']
440 for mean, stdev, param in zip(means, params):
441     print("%f with: %r" % (mean, param))
442
443 ######## Model Definition and Training
444
445 model_bilstm_sa = Sequential([
446   embedding_layer,
447   Bidirectional(LSTM(64, dropout=0.3,return_sequences=True)),
448   SeqSelfAttention(),
449   GlobalMaxPooling1D(),
450   Dropout(0.4) ,
451   Dense(2, activation='softmax')
452 ])
453
454 model_bilstm_sa.summary()
455 model_bilstm_sa.compile(loss='binary_crossentropy',optimizer='adam
      ',metrics=['accuracy'])
456 es = EarlyStopping(monitor='val_loss', mode='min', verbose=2,
      patience=5)
457 mc = ModelCheckpoint('bilstm_sa.h5', monitor='val_accuracy', mode=
      'max', verbose=2, save_best_only=True)
458 history_bilstm_sa=model_bilstm_sa.fit(X_train, Y_train, batch_size
      =1000 ,epochs=200, validation_split=0.1,callbacks=[es, mc])
```

```
best_bilstm_sa = load_model('bilstm_sa.h5',custom_objects={'SeqSelfAttention': SeqSelfAttention})

######## Model Evaluation

score = best_bilstm_sa.evaluate(X_test,Y_test)
plot_training_hist(history_bilstm_sa, title = 'Model BiLSTM with Self Attention')

Y_pred = best_bilstm_sa.predict(X_test)
y_pred = np.argmax(Y_pred, axis = 1)
y_test = np.argmax(Y_test, axis = 1)

print('Model BiLSTM with Self Attention')
print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred,normalize='true')
labels = ["Negative", "Positive"]
display = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
display.plot(cmap = plt.cm.Blues)
display.ax_.set_title("Model BiLSTM with Self Attention",  size=15, y=1.05)

##########################################
## BiLSTM and Transformer Encoder Model ##
##########################################

######## Transformer Encoder layer definition

class TransformerBlock(layers.Layer):
    def __init__(self, embed_dim, num_heads, dense_dim, rate=0.2,**kwargs):
        super(TransformerBlock, self).__init__(**kwargs)
        self.att = MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim, dropout=0.4,kernel_regularizer=L1(l=0.001))
        self.dense = Sequential([
            Dense(dense_dim, activation="relu"),
            Dense(embed_dim),
        ])
        self.layernorm1 = LayerNormalization(epsilon=1e-6)
        self.layernorm2 = LayerNormalization(epsilon=1e-6)
        self.dropout1 = Dropout(rate)
        self.dropout2 = Dropout(rate)

    def call(self, inputs, training):
        sa_output = self.att(inputs, inputs)  # MHA layer
        sa_output = self.dropout1(sa_output, training=training)
        out1 = self.layernorm1(inputs + sa_output)  #
```

```
    normalization
502         dense_output = self.dense(out1)  #Dense layers
503         dense_output = self.dropout2(dense_output, training=
    training)
504         return self.layernorm2(out1 + dense_output)  #
    normalization
505
506 ######## Grid Search for hyperparameters
507
508 def create_model(num_heads, dense_dim):
509     # create model
510     model = Sequential([
511       embedding_layer,
512       Bidirectional(LSTM(100, dropout=0.3,return_sequences=True)),
513       Dropout(0.4),
514       TransformerBlock(200, num_heads, dense_dim),
515       Dropout(0.4) ,
516       GlobalMaxPooling1D(),
517       Dense(2, activation='softmax')
518     ])
519
520     # Compile model
521     model.compile(loss='binary_crossentropy', optimizer='adam',
    metrics=['accuracy'])
522     return model
523
524 model = KerasClassifier(model =create_model, epochs=5, batch_size
    =1000, verbose=0)
525 # define the grid search parameters
526 num_heads = [2,  4]
527 dense_dim = [16, 32, 64]
528 param_grid = dict(model__num_heads = num_heads, model__dense_dim =
     dense_dim)
529 grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs
    =1, cv=3)
530 grid_result = grid.fit(X_train, Y_train,validation_split=0.1)
531 # summarize results
532 print("Best: %f using %s" % (grid_result.best_score_, grid_result.
    best_params_))
533 means = grid_result.cv_results_['mean_test_score']
534 params = grid_result.cv_results_['params']
535 for mean, param in zip(means, params):
536     print("%f with: %r" % (mean, param))
537
538 ######## Model Definition and Training
539
540 num_heads = 2
541 dense_dim = 32 #Firs Dense unit inside the Transformer Block
542
```

```
model_bilstm_trans = Sequential([
  embedding_layer,
  Bidirectional(LSTM(100, dropout=0.3,return_sequences=True)),
  Dropout(0.4),
  TransformerBlock(200, num_heads, dense_dim),
  Dropout(0.4) ,
  GlobalMaxPooling1D(),
  Dense(2, activation='softmax')
])

model_bilstm_trans.summary()
model_bilstm_trans.compile(loss = 'binary_crossentropy', optimizer
    = 'adam',metrics = ['accuracy'])
es = EarlyStopping(monitor = 'val_loss', mode = 'min', verbose =
   2, patience = 5)
mc = ModelCheckpoint('bilstm_tran.h5', monitor = 'val_accuracy',
   mode = 'max', verbose = 2, save_best_only = True)
history_bilstm_trans = model_bilstm_trans.fit(X_train, Y_train,
   batch_size = 1000, epochs = 2000, validation_split = 0.1,
   callbacks = [es, mc])
best_bilstm_trans = load_model('bilsmt_trans.h5',custom_objects={"
   TransformerBlock": TransformerBlock})

######## Model Evaluation

score = best_bilstm_trans.evaluate(X_test,Y_test)
plot_training_hist(history_bilstm_trans, title= 'Model BiLSTM
   Transformer Encoder')

Y_pred = best_bilstm_trans.predict(X_test)
y_pred = np.argmax(Y_pred, axis = 1)
y_test = np.argmax(Y_test, axis = 1)
print('Model BiLSTM with Transformer Encoder')
print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred,normalize = 'true')
labels = ["Negative", "Positive"]
display = ConfusionMatrixDisplay(confusion_matrix=cm,
   display_labels=labels)
display.plot(cmap=plt.cm.Blues)
display.ax_.set_title("Model BiLSTM & Transformer Encoder",  size
   =15, y=1.05)

# Misclassified  Tweets
errors = (y_pred - y_test != 0)
y_pred_errors = y_pred[errors]
y_test_errors = y_test[errors]
x_errors = Test_text[errors]
x_errors.head(30), y_pred_errors[0:30], y_test_errors[0:30]
```

```
583
584 # Wordcloud of each cell of the confusion matrix
585
586 tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
587 tn, fp, fn, tp
588 Results = {"Text": Test_text, "True": y_test, "Pred": y_pred}
589 res = pd.DataFrame(Results)
590 res.head()
591
592 Positive = res[res["True"]==1][["Text","Pred"]]
593 TP = Positive[Positive["Pred"]==1]["Text"]
594 FN = Positive[Positive["Pred"]==0]["Text"]
595 Negative = res[res["True"]==0][["Text","Pred"]]
596 TN = Negative[Negative["Pred"]==0]["Text"]
597 FP = Negative[Negative["Pred"]==1]["Text"]
598 TN.shape, FP.shape, FN.shape, TP.shape
599 TP.head()
600
601 tp_list = TP.values.tolist()
602 tp_text = ' '.join(tp_list)
603 fp_list = FP.values.tolist()
604 fp_text = ' '.join(fp_list)
605 tn_list = TN.values.tolist()
606 tn_text = ' '.join(tn_list)
607 fn_list = FN.values.tolist()
608 fn_text = ' '.join(fn_list)
609
610 def plot_WordCloudsConfusion():
611     wc = WordCloud(background_color = "white", max_words = 50,
    width = 900, height = 500, stopwords = stopwords,colormap="
    autumn")
612     fig,((ax1,ax2),(ax3,ax4)) = plt.subplots(2, 2, figsize =
    (15,8))
613     wc.generate(tn_text)
614     ax1.imshow(wc)
615     ax1.axis("off")
616     ax1.set_title('True Negative', size=15, y=1.05)
617     wc = WordCloud(background_color = "white", max_words = 50,
    width = 900, height = 500, stopwords = stopwords,colormap="gray
    ")
618     wc.generate(fp_text)
619     ax2.imshow(wc, interpolation='bilinear')
620     ax2.axis("off")
621     ax2.set_title('False Positve', size=15, y=1.05)
622     wc = WordCloud(background_color = "white", max_words = 50,
    width = 900, height = 500, stopwords = stopwords,colormap="gray
    ")
623     wc.generate(fn_text)
624     ax3.imshow(wc)
```

```
    ax3.axis("off")
    ax3.set_title('False Negative', size=15, y=1.05)
    wc = WordCloud(background_color = "white", max_words = 50,
   width = 900, height = 500, stopwords = stopwords,colormap="
   summer")
    wc.generate(tp_text)
    ax4.imshow(wc, interpolation='bilinear')
    ax4.axis("off")
    ax4.set_title('True Positive', size=15, y=1.05)

    plt.show()
    plt.close()

plot_WordCloudsConfusion()

################################
###### Models Comparison ######
################################

# Accuracy of the different models
def plot_training_hist():
    fig, ax = plt.subplots(1, 2, figsize=(10,4))
    fig.tight_layout(pad=3.5)
    ax[0].plot(history_cnn.history['accuracy'])
    ax[0].plot(history_bilstm.history['accuracy'])
    ax[0].plot(history_cnn_bilstm.history['accuracy'])
    ax[0].plot(history_bilstm_sa.history['accuracy'])
    ax[0].plot(history_transformer.history['accuracy'])
    ax[0].set_title('Model accuracy on training set')
    ax[0].set_xlabel('epoch')
    ax[0].set_ylabel('accuracy')
    ax[0].legend(['CNN', 'BiLSTM', 'CNN + BiLSMT', 'BiLSTM + SA',
   'Transformer'], loc='best')
    ax[1].plot(history_cnn.history['val_accuracy'])
    ax[1].plot(history_bilstm.history['val_accuracy'])
    ax[1].plot(history_cnn_bilstm.history['val_accuracy'])
    ax[1].plot(history_bilstm_sa.history['val_accuracy'])
    ax[1].plot(history_transformer.history['val_accuracy'])
    ax[1].set_title('Model accuracy on validation set')
    ax[1].set_xlabel('epoch')
    ax[1].set_ylabel('accuracy')
    ax[1].legend(['CNN', 'BiLSTM', 'CNN + BiLSMT', 'BiLSTM + SA',
   'Transformer'], loc='best')
    fig.suptitle('Accuracy comparison', size=16, y=1.07)

plot_training_hist()
```