APPLIED MACHINE LEARNING SYSTEMS II ELEC0132 20/21 REPORT

SN: 16082165

Abstract— Social networks have become a central platform used to share user opinions and emotions. While the public's sentiment has proven to affect various industries such as stock markets, this information is valuable to quantify customer satisfaction and enhance offerings. This paper describes the task of sentiment analysis on movie reviews on a five-scale system. The experiments showed that the simplest LSTM was able to outperform hyper-tuned supervised models. While Neural Networks provide exceptional results in fields such as NLP, they are perceived as black boxes. Thus, this research also aimed to outline a set of techniques used to understand various model designs and the behaviour of their inner layers during training. The best performing model was compared to a state-of-the-art transformer (BERT), which outperformed all other models with a minimal degree of personalisation.

Index Terms— Sentiment Analysis, NLP, Deep Learning, Transformers.

1. INTRODUCTION

Since the emergence of mobile devices and interconnected systems, data volume, velocity and variety have not ceased to increase dramatically. This rapid proliferation has allowed corporations to achieve operational excellence and provide enhanced experiences through tailored recommendations [1]. The rapid growth of social media has created opportunities to generate patterns from user interactions and comprehend their opinions towards specific topics at unprecedented scales. Nevertheless, the unstructured nature of user-generated data mandates new computational techniques for social media mining. In fact, Natural Language Processing tasks have been initially based on shallow models such as Logistic regression and trained on hand-crafted sparse features [2]. More recently, Deep learning architectures have demonstrated their ability to solve complex NLP tasks such as named-entity recognition and text classification.

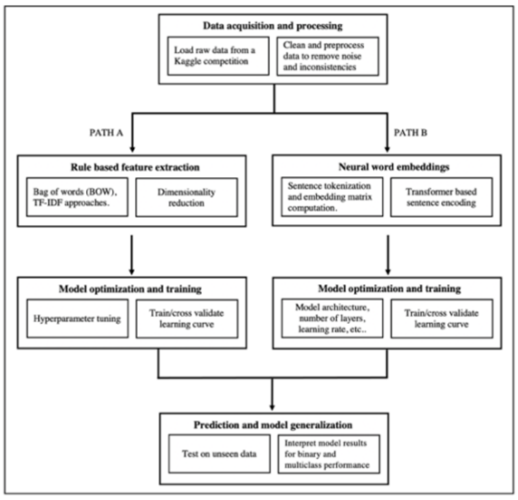

Data acquisition and processing
Load raw data from a Kaggle competition
Clean and preprocess data to remove noise and inconsistencies
PATH A
PATH B
Rule based feature extraction
Bag of words (BOW), TF-IDF approaches.
Dimensionality reduction
Neural word embeddings
Sentence tokenization and embedding matrix computation.
Transformer based sentence encoding
Model optimization and training
Hyperparameter tuning
Train/cross validate learning curve
Model optimization and training
Model architecture, number of layers, learning rate, etc..
Train/cross validate learning curve
Prediction and model generalization
Test on unseen data
Interpret model results for binary and multiclass performance

Fig. 1 *ML pipeline for Supervised and Deep Learning sentiment classification*

These models can achieve multi-level feature representation learning and have revolutionised the field of text processing through the creation of neural word embeddings. Commonly, fully connected deep learning models are not suited for processing textual representations due to the huge number of parameters they require [3]. Thus, the aim of this report is to describe the state-of-the-art techniques used to solve the task of sentiment classification using a movie review dataset. To achieve this objective, the concepts of word vectorisation and embeddings will be first explored through the use of hand-crafted feature extraction methods (Fig1, Path A). In order to understand the profound differences between these two approaches, an analogy will be provided by solving the same task following a deep learning approach (Fig1, Path B). First, review data will be processed in order to remove textual noise. Techniques such as Bag of words (BOW) or TF-IDF will be compared during the feature extraction stage. Due to their sparse nature, the resulting vectors will be transformed using dimensionality reduction. Various methods will be discussed to outline the process of dealing with high feature spaces and reducing noise for improved learning. Once pre-processed, train, validation and test sets will be used to classify labels using Logistic regression. The performance of this baseline model will be compared to results obtained using neural word embeddings and Neural Network architectures. Finally, this paper will describe advanced techniques used to overcome various RNN limitations. Discussions will focus on memory-augmenting strategies such Attention and the role of this mechanism in state-of-the-art transformer models.

2. LITERATURE SURVEY

2.1. Natural language processing and sentiment analysis

The field of Natural language processing (NLP) is a subset of Artificial intelligence (AI) which defines different types of interaction between computers and humans using the natural language. Initially, solving NLP problems typically involved large sets of complex hand-written rules. This methodology called "Rule-based" NLP, has demonstrated to be relatively ineffective and time consuming. The development of machine learning (ML) has revolutionised the NLP field by leveraging statistical inference to automatically derive rules from the analysis of large corpora. Usually, solving complex NLP tasks through machine learning encompasses the use of two main approaches. On the one hand, unsupervised learning aims to establish a relationship between unlabelled data points in order to achieve tasks such as clustering or dimensionality reduction. On the other hand, supervised learning establishes a relationship between different features and their predefined labels [4]. This subset includes classification, where classes are discrete. More recently, the area of sentiment analysis has received important amounts of attention. Also called opinion mining, this field deals with identifying opinion patterns from textual data such as social network feeds.

The outcomes of sentiment can be represented in the form of a binary or a multi-class problem given a scale of likeness [5]. Attaining accurate classification depends on the ability of ML models to fit large amounts of samples and generalise learning on unseen data. Although choosing the right model can impact overall accuracy, the biggest challenge is to train models with informative features that lead to low generalization error.

2.2. Feature extraction overview

In the context of text processing, feature extraction defines the process of encoding data into vector representations in order to extract salient features [6]. Statistical NLP has surfaced as the primary option for modelling text in a numerical way. This approach has revealed to suffer from various limitations such as the curse of dimensionality while learning joint probability functions of large datasets [7]. Hence, these limitations have created the need to establish low dimensional and distributed word representations.

2.2.1 Static word embeddings: BoW and TF-IDF

The simplest method to encode a corpus is called Bag of words (BoW). In this model, sentences (called documents) are modelled as flat vectors where each word represents a feature. While BoW disregards grammar and word order, this method is based on the principle of multiplicity which assumes that documents are similar if they have similar content [8].

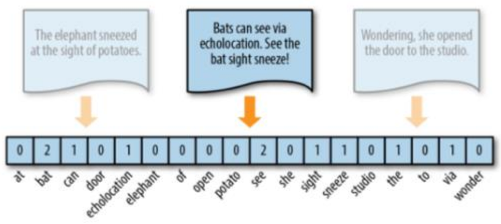

The elephant sneezed at the sight of potatoes.
Bats can see via echolocation. See the bat sight sneeze!
Wondering, she opened the door to the studio.
0 2 1 0 1 0 0 0 0 2 0 1 1 0 1 0 1 0
at bat can door echolocation elephant of open potato see she sight sneeze studio the to via wonder

Fig. 2 *Token frequency as vector encoding [9]*

BOW has proven to deliver reasonable results for sentiment analysis, as words associated to specific classes generally overwhelm others towards a certain polarity [10]. However, scoring word frequency without considering their semantic meaning can lead to erroneous representations. Indeed, highly represented words can adversely dominate documents without necessarily containing informational content. Thus, rarer but perhaps domain specific words can be discarded by the model, leading to misclassification. TF-IDF (Term Frequency-Inverse Document Frequency) is an amelioration of the BoW model (Fig 2). This approach measures the number of times a word appears in a given document (TF) while penalising those that are frequent across all documents (IDF) as following:

$$w_{i,j} = tf_{i,j} * \log\left(\frac{N}{df_i}\right) \qquad (1.1)$$

Here $tf_{i,j}$ refers to the number of times i appears in j, df_i to the number of sentences containing i and N to the total number of sentences. However, this approach generally results in the creation of vectors with important amounts of zero values. These sparse vectors demand more computing resources and grow exponentially with the vocabulary size.

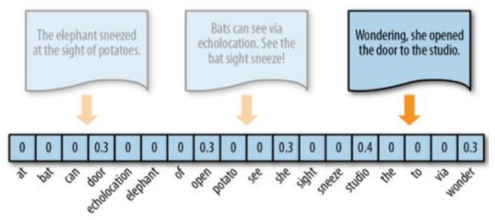

The elephant sneezed at the sight of potatoes.
Bats can see via echolocation. See the bat sight sneeze!
Wondering, she opened the door to the studio.
0 0 0 0.3 0 0 0 0.3 0 0 0.3 0 0 0.4 0 0 0 0.3
at bat can door echolocation elephant of open potato see she sight sneeze studio the to via wonder

Fig. 2 *TF-IDF vector encoding [9]*

A simple approach consists of applying simple data cleaning such as removing punctuation or extremely frequent words. Alternatively, vocabulary can be defined based on group of words called "grams", which allows to capture more meaning from a document. For example, creating a vocabulary of two-word pairs is a bigram model [11].

2.2.2 Dealing with high dimensional spaces

Although the aforementioned approaches can dramatically reduce vocabulary size, large vectors obtained through static word embedding require the use of more systematic practices. Principal component analysis (PCA) is commonly used for dimensionality reduction and aims to find linear combinations that capture maximal variance between variables [12]. In text processing, PCA is achieved through the singular value decomposition (SVD) of a *(n × d)* matrix ***A*** where *n* and *d* are the vocabulary size and the vector dimensions. The resulting principal components represents the product of both the left singular vector matrix and singular value matrix, whereas V^T are the loadings of those principal components such that:

$$A = U_j \Sigma V_i^T \qquad (1.2)$$

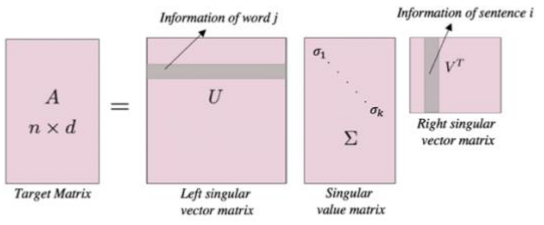

Information of word j
Information of sentence i
A
n × d
=
U
σ1
σk
Σ
V^T
Target Matrix
Left singular vector matrix
Singular value matrix
Right singular vector matrix

Fig. 3 *SVD decomposition for a matrix A [12]*

However, PCA relies on the fact that principal components are a linear combination of all *d* variables and that the loadings are typically nonzero. Consequently, the sparse nature of the vectors obtained through static word embedding does not allow the use of this approach. Other methods such as Latent semantic analysis (LSA) have shown to be more adapted to NLP tasks. LSA leverages the principle of truncated singular value decomposition (TSVD) as a rank lowering method to reduce the original vector space into *k* dimensions (Fig 4):

$$A = A^{n\,x\,d} \cong A_k{}^{n\,x\,d} = U_i{}^{n\,x\,k} . \Sigma^{k\,x\,k} . V_i{}^{k\,x\,d} \qquad (1.3)$$

Adopting this approach does not only allow to achieve the benefit of computational simplification, but leverage words semantic meaning to reduce noise and eliminate words that do not contribute to the meaning of a document [13].

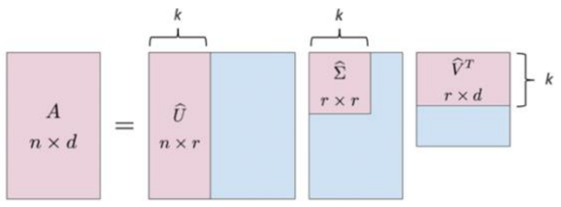


Fig. 4 *TSVD decomposition for a matrix A [12]*

2.2.3 Intuition on Deep learning and Neural networks

Neural network architectures are composed of a set of units called neurons that are categorised into three types of layers. Generally, the input layer takes a series of inputs $\{x_1, \ldots, x_k\}$ and corresponding weights $\{w_1, \ldots, w_k\}$. During forward propagation, inputs are passed into a hidden layer where an activation function is applied to produce a scalar output *y*. Using an error function, the network compares the predicted values and the actual outputs and recalculates weights using stochastic gradient descent. NN's differ from traditional machine learning by their ability to leverage this mechanism to achieve feature learning during training.

2.2.2 Semantic word embeddings: Word2Vec

The previous section has described the process of generating word vector representations using static word embeddings. Approaches such as TF-IDF have revealed to be relatively ineffective due to the sparse vectors they generate. Despite the use of n-gram models and LSA, these vectors are not capable of capturing the relationship between words within different documents, nor their semantic meaning. This limitation is due to the fact that these representations exclusively contain non-negative elements. Thus, vectors with a cosine distance of 1 are considered far apart, even if they are semantically similar. The development of Deep learning has enabled the creation of new encoding techniques called semantic embeddings. Although various algorithms can be adopted, this paper will only focus on Word2Vec due to page limitations. Created by

Google, Word2Vec aims to encode text along a continuous scale with a distributed representation. Therefore, documents are represented in a feature space that has been embedded to represent word similarity as opposed to word occurrence [9]. This algorithm can learn word representations based on either a continuous bag-of-words or a skip-gram model.

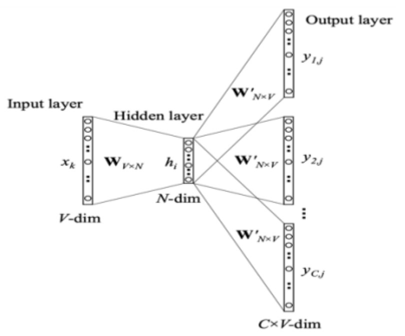


Fig. 5 *Skip-gram model architecture [14]*

Although both methods rely on the same principles, skip grams have proven to perform better, especially on large corpora. The training objective of the Skip-gram model is to find word representations that predict surrounding words in a sentence [15]. Since Word2Vec is an unsupervised approach, data is automatically sampled at the beginning of the training phase (Appendix A). As illustrated in Figure 5, this model takes a sequence of vectors $\{x_1, \dots, x_k\}$ in its input layer and aims to maximise the average log probability:

$$\frac{1}{T}\sum_{k=1}^{T}\sum_{j}\log p\,(x_{k+j} \mid x_k) \qquad (1.4)$$

As skip-grams aim to classify a set of classes C, they produce various different multinomial distributions at the output layer:

$$p(w_{c,j} = w_O \mid w_I) = y_{C,j} = \frac{\exp(u_{C,j})}{\sum_{j'=1}^{V}\exp(u_{j'})} \qquad (1.5)$$

Since output layer panels share the same weights:

$$u_{C,j} = {v'_{wj}}^{T} v_{wI} \;\text{ and }\; u_{j'} = {v'_{wj}}^{T} v_{wI} \qquad (1.6)$$

Although it does not involve dense matrix multiplications, this formulation is impractical because the cost of computing $\nabla p(w_{c,j} = w_O \mid w_I)$ is proportional to the vocabulary size V. In fact, training a Word2Vec model with word vectors having 300 components on a vocabulary size of 10000 would result in generating 3 million weight matrices at the input and output layers. To overcome this constraint, Mikolov et al. proposed an alternative method: Negative sampling. This approach is derived from Noise Contrastive Estimation (NCE) which argues that good models should be able to differentiate data from noise by means of logistic regression. Thus, negative sampling consists of defining a supervised learning problem that aims to distinguish a target word with generated noise (negative samples). Adopting this approach allows to update solely a sample of the training weights during each iteration. Negative samples are used to replace $p(w_O \mid w_I)$ in the skip-gram objective and are drawn from the noise distribution $P_n(w)$ as following:

$$log\ \sigma\left({v'_{w0}}^{T} v_{wI}\right) + \sum_{i=1}^{k} \mathbb{E}w_i \sim P_n(w)\left[log\ \sigma\left(-{v'_{wi}}^{T} v_{wI}\right)\right]$$

2.3 Dealing with imbalanced datasets

Working with real data encompasses a variety of challenges. In a majority of cases, large amounts of data are generated with a skewed distribution. In extreme cases, class imbalance results in the incapability of classifiers to discriminate poorly represented classes and leads to overfitting. While there is no generic approach to deal with this anomaly, text augmentation techniques are used to generate synthetic instances of an underrepresented class. Lexical substitution is a method that replaces words without changing the meaning of a document. Depending on available computational resources, substitution could be achieved using the semantic embeddings described previously, which rely on the cosine distance between vectors to find synonyms. Alternatively, Garg. et al. have shown the effectiveness of generating adversarial examples by utilising masked language models such as BERT [15]. This method consists of randomly replacing words in documents by masks and generating text variations using these mask predictions. In their paper, Xie et al. have demonstrated the use of these models to augment text through an alternative technique called back translation [17]. This approach aims to translate documents and reconstruct the output by converting it to the original form. Although the aforementioned solutions have proven to deliver satisfactory results, a multitude of other approaches can be adopted and were not described due to page limitations.

3. DESCRIPTION OF MODELS

3.1 Introduction to Recurrent Neural Networks

Given the sequential nature of sentences, recurrent neural networks (RNN) are one of the most popular architectures used in NLP. As illustrated in Fig 6, RNN's differ from feed forward networks by their ability to use previous outputs to be used as inputs at a later time step *t* [15].

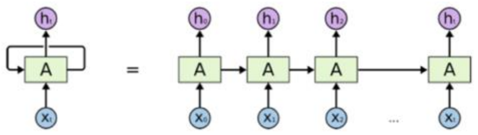


Fig. 6 *Architecture of a simple Recurrent Neural network*

These "long-term dependencies" h_t are called hidden state and are obtained using of an activation function as following:

$$h_t = f(U_{xt} + W_{h_{t-1}}) \qquad (1.7)$$

Because feedback loops occur at every time step, each hidden state does not only contain traces of the previous hidden state, but also those that preceded h_{t-1}. Thus, this type of networks relies on a different type of back propagation called BPTT (backpropagation through time). The gradients accumulation can result in obtaining very unstable networks leading to the exploding or vanishing gradient problem. Mathematically, h_t can be approximated by ignoring inputs x_t and express its weights Was a diagonal matrix of eigen values so that:

$$\left\{\begin{matrix} h_t = W_{h_{t-1}} = (W^T)h_0 \; and \; W = A\, diag(\lambda) A^{-1} \\ \Leftrightarrow \\ h_t = A\, diag(\lambda) A^{-1} h_0 \end{matrix}\right\} \qquad (1.8)$$

As a result, any eigenvalue approaching 0 will make gradients shrinking exponentially preventing the model from learning any pattern from previous sequences. On the other hand, values greater than one would lead the gradients of long-term interactions to be exponentially smaller than the gradient of a short-term interaction making the learning extremely slow or leading to a Nan loss value during training [18].

3.2 Long-Short-term-Memory

Long-Short-Term-Memory (LSTM) is a special type of RNN, able to learn more resourcefully long-term dependencies. LSTM models filter the amount of information through a cell state C_t. These cells are regulated by various gates throughout the learning process.

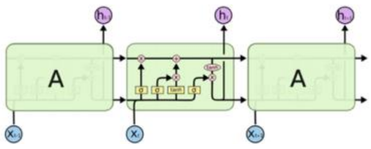


Fig. *7 Architecture of LSTM*

During each iteration, a sigmoid layer f_t (Forget gate layer) takes a linear transformation of concatenated previous hidden states h_{t-1} and an input x_t. Depending on the output (0 or 1), the model decides to completely remove or keep information in the dimension. Next, a tanh layer creates a vector $\tilde{C}_t$ of new candidate values that could be added to the state. Input gates i_t decides how much values of $\tilde{C}_t$ must be combined into the current cell state. The final operation consists of updating the previous cell state C_{t-1} by using the input and forget gates with new candidate cell states so that:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \qquad (1.9)$$

Although forget gates still rely on multiplication, LSTM's solves the vanishing / exploding gradient problem by simply adding the current state with a new input, thus preserving a constant error during backpropagation [15]. Nevertheless, LSTM models generally suffer from the inability to capture context from long sentences, as the probability of keeping meaning from two words is inversely proportional to their distance. In addition, this model suffers from a relative lack of parallelisation which makes them relatively slow to train.

3.4 Convolutional Neural Networks

CNN's are feed forward NN's that enable processing inputs in a parallel and distributed way. They take a document as a matrix $A \in R^{V\text{x}D}$, where V symbolises the document length and D is the embedding dimensions (Fig 8).

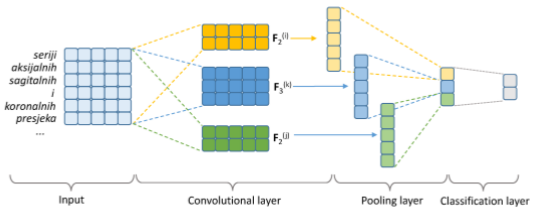


Fig. *8 Architecture of a CNN for text classification*

The dimension of the input document depends on whether input vectors are initialised randomly or using pre-trained word embeddings such as Word2Vec. These inputs are fed to a convolutional layer which is a linear operation followed by a non-linear transformation. Linear operations typically consist of multiplying elementwise the result of a 1D window applied over the input by a filter. Depending on the chosen stride n, the filter slides over the input to extract n-gram features called receptive fields F_i [19]. These fields allow CNNs to recognize complex patterns by combining lower-level elementary features into higher-level features. These fields allow CNNs to recognize complex patterns by combining lower-level features into higher-level features. Then a non-linear activation function f (typically ReLU) is applied elementwise to F_i, resulting in the creation of a feature map associated with the filter:

$$c_i = f(F_i) + b \qquad (2.0)$$

For tasks such as sentiment analysis, the exact positions of the features in the input document do not matter. Instead, the aim is to identify whether certain features are present or absent to determine a comment's polarity. Thus, pooling methods such as *k-max pooling* to extract the *k* greatest values from each of the feature maps and concatenate them to obtain a final vector. Depending on the classification task (binary or multiclass), a final activation function is applied to output the probability distribution for each class. For binary tasks SoftMax would be used so that:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \qquad (2.1)$$

Through their parallel architecture, CNN's are able to solve the aforementioned LSTM computational limitations. While they have proven to deliver satisfactory results for short sentence classification, they suffer from the inability to capture long term dependencies in larger inputs.

3.4 Transformers Architectures and Attention

Introduced by Vaswani et al in their paper "Attention is all you need", Transformers are considered as the new state of the art model for NLP (Appendix B). Transformers are able to overcome RNN's limitations when processing long sentences by leveraging the Attention mechanism.

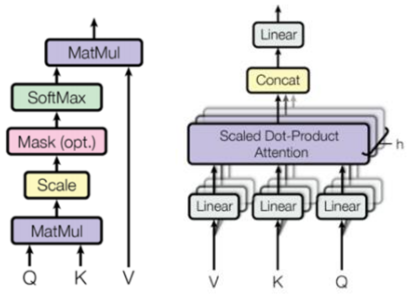


Fig. *9 Scaled Dot-Product vs Multiheaded Attention*

As outlined above, attention takes three main inputs at the encoder level. First, Q represents a matrix of dimension d_k that contains serval queries. A query is typically a word vector representation that is usually pre-defined and tuned during training iterations. The matrix K is a representation of all the word keys within a specific sentence and is associated with V which encodes the values of each word. Unlike traditional RNN's which uses previous hidden states h_{t-1} and inputs x_t, transformers process words concurrently. Since the meaning of words depends on their position in sequences, positional encoding is used to capture context-sensitive representations. This data is included in each representation to generate attention weights a through the matrix product QK^T. Through this process, weights are defined depending on how each word Q is influenced by all other words of the sequence (Appendix C):

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad (2.2)$$

This information allows to mask out the associated values that are not relevant to the query and helps to generate better word representation compared to RNN's. Following this principle, Vaswani et al. proposed an alternative form of attention called Multi-head Attention in order to achieve the parallelisation that is lacking to recurrent models. Instead of performing one attention for each word, it is linearly projecting Q, K and V into trainable matrices W_q, W_k, W_v to generate a number of h attentions. This operation results in the creation of low dimensional vectors that are transformed and concatenated using a final linear transformation:

$$Multihead(Q, K, V) = Concat(head_1, \dots, head_h)W^O$$

Multi-head attention enables to jointly attend information from different representation subspaces while benefiting from the reduced dimensions of each head to keep the same computational cost as self-attention [20].

3.4.1 BERT and transfer learning

The implementation of attention-based transformers was first designed to process data using an encoding-decoding process. While sequence-to-sequence models have proven to perform efficiently for tasks such as text translation, they are not really adapted for solving classification problems. The recent introduction of BERT (Bidirectional Encoder Representations from Transformers) has brought significant improvements in the field of text classification. Different variants such as bidirectional LSTM's have enabled to learn more accurate representations. This approach captures the meaning of sentences separately form right to left then left to right and concatenates it to form a single entity. The main limit behind this method is that concatenation leads to slightly losing the true context between words.

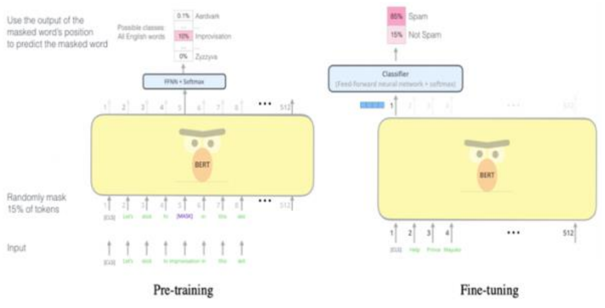

Use the output of the masked word's position to predict the masked word
Possible classes: All English words
0.1% Aardvark
10% Improvisation
0% Zyzzyva
FFNN + Softmax
BERT
Randomly mask 15% of tokens
[MASK]
Input
Pre-training
85% Spam
15% Not Spam
Classifier
BERT
512
Fine-tuning

Fig. 10 Bert training and fine tuning for spam classification

BERT overcomes this limitation by randomly masking a percentage of the input tokens. Then, the model builds a bidirectional representation of sentences by predicting back masks through its stacked encoders. The loss is defined by the extent to which the model predicts masked words, as opposed to the whole sequence reconstruction error. This process is known as Masked LM (Language model) and happens during the pre-training phase. Since many downstream tasks such as question answering are based on the relationship between sequences, BERT's pre-training phase employs next sentence prediction to achieve optimal learning. Because this paper mainly focuses on sentiment analysis, the aforementioned notion was just mentioned for reference and will not be described further.

Once pre-trained, the final step is to fine-tune the model on any downstream task. This step simply consists of adding a shallow classifier or a series of dense layers to the pre-trained BERT layer and feed the model with task-specific data and their corresponding labels [23].

4. IMPLEMENTATION

As stated in the introduction, this paper aims to provide a comprehensive understanding of state-of-the-art methods used to solve NLP tasks, more precisely sentiment analysis. This section will describe series of experiments involving several models and architectures variations. The motivation behind this approach is to support the knowledge acquired in the literature review through empirical evidence and deliver an intuition about their practical implementation.

4.1 Data Loading and Pre-processing

First the functions *load_train_data()* and *load_sub_data()* were used to load the dataset and the unlabelled submission test set. These functions leverage the Pandas library to convert the original *tsv* files into a structured DataFrame, from which irrelevant columns were dropped such as sentence ID's. This operation resulted in the obtention of a dataframe containing 156 060 user reviews and their corresponding labels. Thus, this task consisted of classifying reviews into five different classes which is normally defined as "Fine grained sentiment analysis". As part of data pre-processing, labels were encoded using the *sklearn labelencoder* module. In addition, the function *data_cleaning()* was used to remove noise from each sentences such as special characters, URL's, HTML tags and punctuation. This function also removed outliers such as extremely rare and too common words that could bias the classifier's decision boundaries.

4.2 Data partitioning

As data was loaded from a Kaggle competition, the test set provided did not include labels. Typically, competitions rank users based on test label submissions. This approach suffers from the limitation that rankings are only based on accuracy. Although this measure assesses performance on a high level, accuracy does not provide enough information to determine whether predictions can be generalised. As the submission data contained 55810 instances, the original train data was split into training (70%), validation (15%) and test (15%) sets. The repartition of validation and test data was designed to obtain equally sized portions (23409 points) that were large enough to assume that the observed model behaviour would be consistent with the accuracy observed after submission. While this approach reduces the size of the train set, it provides more granular information about model behaviour such as potential overfitting. These insights are described in the next section through result tables (Precision, recall, f1-score) and visual illustrations such as learning curves and confusion matrices.

4.3 Sentiment analysis: Supervised learning approach

This section depicts the practical execution of numerous feature extraction methods outlined in the literature survey. More precisely, experiments compare the results obtained through the use of Logistic regression (LR) using both a static word embedding (TF-IDF) and semantic embedding method (Word2Vec). The motivation behind this approach is to demonstrate the main findings of the literature survey. Namely, that semantic word embeddings enable enhanced classification results through their ability to capture more context in sentences. Logistic regression was selected for its capacity to address multiclass problems with minimal code changes and rapid training time. These tests also allowed to establish a baseline to compare the performance of basic supervised methods with state-of-the-art models.

4.3.1 Static word embeddings

The first experiment consisted of creating a basic classifier based on TF-IDF features. Features were first converted using the *TfidfVectorizer* module available through *sklearn*. This library allowed to convert the initial collection of raw documents to a matrix of TF-IDF features. In order to avoid losing relevant information, the shape of the resulting array was not defined as a parameter. Thus, the feature matrix was only created depending on the top max_features ordered by term frequency across the corpus [21]. As mentioned in the literature survey, TF-IDF features can be created following a n-gram model. Thus, the function *return_best_tfidf ()* was used to identify the n-gram parameter. This function uses *sklearn* pipeline and grid-search modules to iterate through a list of n-gram parameters and identify the configuration that would lead to the best results during inference.

4.3.2 Dimensionality reduction and Inference

While the aforementioned approach allowed to capture all corpus information during feature extraction, it resulted in the obtention of a highly dimensional sparse matrix of shape (156060, 16056). Sparse matrices usually affect classification performance due to the large amount of information they contain and are extremely slow to train [9]. Consequently, TSVD was applied as a dimensionality reduction technique through *sklearn* (aka LSA). This approach was adopted to allow faster training but also to investigate if LSA could counterbalance the inability of TF-IDF to extract the semantic meaning of words for superior predictions (Appendix E).

4.3.3 Intuition on dynamic embeddings

Although dynamic word embeddings mainly leverage deep learning, it was interesting to compare the performance of the baseline model when trained separately on static and dynamic feature embeddings. For simplicity a pre-trained word2vec model was loaded through *word2Vec_transform()* using the *spacy* library. This model was used to convert tokens within sentences into fixed length (300) word vectors that were used during training. The results obtained through all supervised methods are discussed in the next section.

4.4 Sentiment analysis: Deep Learning methodology

The results of the previous experiments revealed that the adoption of word2vec provided enhanced results compared to static word embeddings. Hence, the next step consisted of implementing the latter approach following a Deep learning approach. This section summarises the execution of three main models using TensorFlow, namely LSTM, CNN and BERT. Throughout series of tests, the architecture of these models will be redefined following two approaches. The first one is to build a baseline model having the simplest architecture as possible, then analyse its behaviour as its complexity increases. The second approach relies on the principle of ablation studies. This methodology examines the performance of an AI system by removing components of its architecture [24]. This approach extremely useful to identify the contribution of specific components to the overall system. In order to identify model bottlenecks and analyse the behaviour of each layer, visualisations will be generated on Tensorboard using TensorFlow. Thus, training models was achieved using the pre-built *"fit"* method as opposed to a custom training loop. This approach employed callbacks to obtain weight histograms, training curves and confusion matrices during each training iteration.

4.4.1 Neural word embeddings

Although experiments involved implementing models with different architectures, they were all initialised using the same embedding layer. This element was built using the function *load_pretrained_embedding_model()* which uses genism library to create the word2vec word embeddings. This function also transforms data by padding inputs in order to obtain fixed length vectors. While this function leverages embeddings pre-trained on large corpora, the alternative of training word2vec from the ground up specifically on the train data was also explored through *train_Word2Vec_model().* Finally, the last alternative that was tested was to set the "trainable" parameter of the embedding layer as True. The outcome of these methods is described in the next section.

4.4.2 LSTM experiments

The LSTM implementation was based on the first approach mentioned previously. Its initial version is a basic model composed of one LSTM layer followed by a dense layer. This model allowed to have a deep learning baseline and compare its effectiveness with results obtained through supervised learning. In addition, this version allowed to determine the word embedding approach that would lead to the best results. It also helped determining if using a bi-LSTM would enable getting better sentence representations and thus obtain better results. Model complexity was increased by adding a series of dense and LSTM layers. Following the methodology adopted by Chieh-Chi et al. in [25], both "pooling on feature" and "pooling on prediction" strategies were investigated. The difference between these two approaches exists in the position of the pooling layer (Appendix D). In addition, TensorFlow allows to return hidden states and outputs from specific LSTM layers. Thus, this method was also experimented by setting the parameter *return_sequence* to True when aggregating different layers.

4.4.3 CNN experiments

In order to study the behaviour of CNN's for this task, an ablation study was conducted. The chosen architecture was inspired from C.Ruiz and S.Bedmar's paper [26]. It consisted of employing several convolution layers where a set of filters are sliding along the word embedding matrix of each review to generate feature maps of each data point. The proposed architecture applied a series of convolutional layers, each comprising different kernel sizes to capture meaning from different n-gram representations and followed by a Global-Max-Pooling. ReLU was used as the main activation function to avoid the problem of gradient vanishing [27]. All the resulting feature maps were then concatenated and passed to the final output layer. The ablation study consisted of analysing model performance when reducing gradually the number of convolution layers. Adding additional dense layers after concatenation was also explored.

4.4.4 Hybrid approach

The last model was also inspired from C.Ruiz and S.Bedmar's work [26]. This architecture leverages both the benefits of LSTM's and CNN's through a hybrid model. The motivation behind this approach was to extract local features from the LSTM layer output, which corresponds to the sequence of hidden states return. Convolutional layers are then used to extract local information and aggregate the most important features using Global-Max-Pooling. The resulting operation is finally concatenated and passed to the final output layer for class prediction (Appendix F). Please note last two models were implemented using Keras API instead of the *Sequential* library as it allowed to achieve higher degree of parallelization and achieve faster training.

4.4.5 State of the art transformers: BERT

As a conclusion to the aforementioned experiments, the task of sentiment analysis was run using a final model: BERT. The motivation of employing this approach was to demonstrate the ability of transformer architectures to achieve state of the art results compared to the manually built best performing model. The proposed architecture was purposely kept simple in order to prove that BERT is able to produce superior results without any advanced or task-specific architecture. As outlined in the previous section, data was transformed using the function *bert_encode()* which uses Keras tokenizer to obtain three main representations. Token embeddings were used to identify the beginning (CLS) and end of sequences (SEP). Similarly, the segment and positional embeddings were returned to allow the model to identify masked words and their position (Figure

11). These the three inputs were then fed into a pre-trained BERT layer for fine-tuning. Following the work of Srivastava et al., the use of a certain number of dense layers and Dropout were investigated in order to reduce overfitting before the output layer [29].

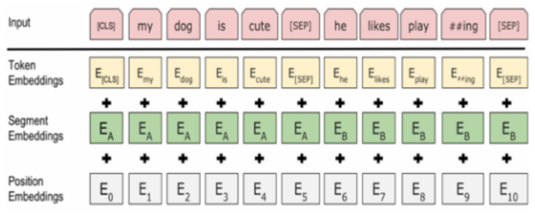


Fig. 11 Bert sequence representation encoding

4.5 Model selection and hyperparameter tuning

Following the previously mentioned experiments, the best performing model was fine-tuned in order to deliver a robust analysis. To achieve this aim, more systematic techniques were used to optimal results. At first, the given model's parameters were tuned using a manually built grid search through *hypertune_model(),* built from the *hp_params* module of TensorFlow. The choice of uniquely fine tuning the best performing model is justified by the lack of computational resource and the large amount of time required to achieve this operation for each model. Once the best version of the model returned, it was tested using additional synthetic data that was created using *back_translation_augmentation()*. This function uses a mixture of backtranslation and embedding synonym replacements to achieve data augmentation.

5. EXPERIMENTAL RESULTS AND ANALYSIS

This section will summarise the results obtained through the experiments mentioned above. The aim of this methodology is to prove that the simplest deep learning model can achieve superior performance compared to the supervised baseline. The different subsections will describe the thought process behind the incremental changes applied to different models and their impact on performance. While changes were made following an analytical methodology (as opposed to random empirical test), the main motivation behind these experiments was not to achieve the best results possible for each model. Instead, their aim was to understand the main differences between various designs and understand how architectural changes affect their performance.

5.1 Performance metrics and visualisations

Usually, the performance of models on binary classification is measured through AUC (Area under the curve). Since the fine-grained sentiment analysis involved the classification of five classes, an alternative approach was adopted. In fact, accuracy was used as a preliminary measure of performance when comparing different models or architectures. While this measure indicates the proportion of true results among the total number of cases examined, models can potentially display high accuracy while only recognising very small portions of a given class. In addition, high accuracy can happen due to inequal class distribution. In that case, models are biased toward a certain class and fail to predict correctly underrepresented classes. Therefore, precision and recall were analysed after each experiment to validate whether accuracy was truly representing model performance. Since F1-macro represents the weighted average of the precision and recall, this measure was used as the final measure for decision making. A large value indicates that a classifier performs well for each individual class. Thus, it represents the most suitable measure when measuring performance on imbalanced data [30]. In addition, tendencies of models to overfit were also visualised using learning curves. Although these metrics allow to picture overall performance, the behaviour of each individual layers between each iteration usually represents a backbox and is extremely difficult to interpret. Therefore, the weight and bias distribution were used to capture more meaningful information and achieve robust results through ablation studies.

5.2 Supervised Learning experiments

Generally, the results observed were in accordance with the main findings of the literature survey. predictably, the LR model delivered the lowest performance when used with basic TF-IDF features. The model returned a relatively low F1-score (0.15), which was mainly due to its inability to recognise comments with very negative and very positive polarities. After applying LSA, a significant improvement was observed both in terms of accuracy (which increased to 0.55) and F-1 score (which raised to 0.33). In a similar way, the use of word2Vec features provided an enhanced result. While the macro F-1 score increased to 0.46, this approach allowed to largely increase accuracy to 0.63.

5.3 LSTM and bidirectional variants

The comparison of various supervised approaches allowed to establish that using LR in conjunction to word2Vec was the best approach to tackle sentiment classification. This section consisted of verifying that even the most basic Deep learning implementations are able to outperform the aforementioned baseline model. Hence, the first experiment consisted of using a single layer LSTM (Appendix G) model to classify user reviews. The first model iteration consisted of feeding the input layer with manually trained word embeddings (Appendix H). The resulting model confirmed the previous assumption and delivered superior results compared to the baseline LR. For instance, the classification report showed an increase in accuracy (0.65) and more importantly in F1-score (0.53). These results suggest that the simple LSTM was able to classify more efficiently classes, but also to appropriately identify true positives. The main motivation behind training manually the word2Vec model was to build word embedding vectors on this specific data.

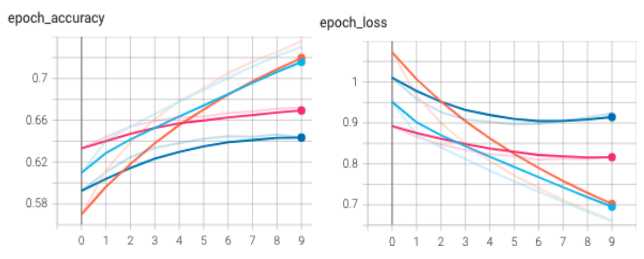


***Fig. 12**: Learning curves for tests with three different embeddings*

However, main limitation of this method is that the vocabulary is only built on task specific data. Consequently, the resulting model might build embeddings based on too specific vocabularies and fail to capture the meaning of words in a general context. Thus, the performance of LSTM was evaluated using a pre-trained word2Vec model. The latter approach revealed to allow more accurate predictions. More precisely, the model achieved an overall accuracy of 0.67 and even improved its ability to distinguish between classes as recall increased to 0.57. This trend was also visualised on TensorBoad by plotting the two models' learning curves (Fig 12). While the training curves looked similar the train data (light blue and orange), it can be observed that the pink curve corresponding to the pre- trained word2Vec achieved better validation results both in terms of accuracy and loss. To benefit from the advantages of both approaches, a last experiment consisted of using the same settings as the best performing LSTM while allowing its input layer to train the embedding model during each iteration. Nevertheless, this approach did not allow to increase model the performance. Therefore, it will not be discussed in this report.

While the objective of using a simple LSTM was to identify the best word embedding strategy, the simplest version of this model also revealed to perform better than any of the previous supervised techniques. Nonetheless, the learning curve visualisations suggest that the model can be further perfected. Therefore, each LSTM layer was followed by a dense layer. In order to summarise the obtained output, a pooling layer followed by another dense layer was added before the final output layer. Basic empirical trials revealed that Global-max-pooling revealed to be the best approach. These tests also showed that "pooling on prediction" (Dense layer preceding pooling) attained superior results. Taking these factors into account, the model with the best results consisted of stacking consecutively a series of 3 LSTM-Dense layers. The resulting model (Appendix I) allowed to rise overall accuracy to 0.68 and recall to 0.58. However, adding additional LSTM layers did not seem to contribute to improve the model substantially and led to a precision that was less equally distributed among classes (Appendix J). Moreover, Tensorboard visualisations revealed that the last dense layer's weight histograms did not change in terms of shape. One explanation could be that this layer simply did not participate in training. This hypothesis was confirmed as removing the layer did not affect model performance.

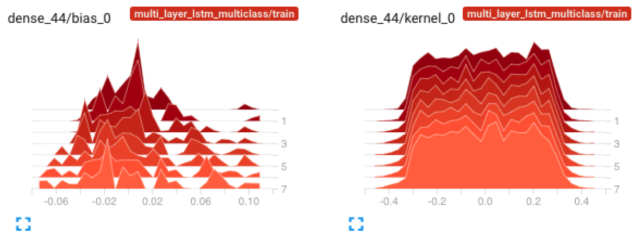


Fig. 13: *Weight histograms for the last Dense layer*

5.4 Convolutional Neural Networks

While the previous section consisted of adding components to increase gradually model performance, the study of CNN architecture followed an ablation study approach. The initial model consisted of 4 components, each containing a 1-D convolutional layer, a max pooling layer and a dense layer.

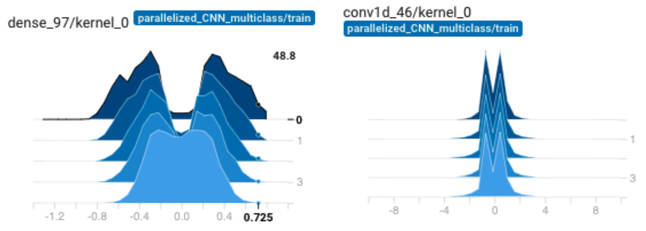


Fig. 14: *Weight histograms for third 1D-Conv layer*

Each convolution captured different word representations. Namely in the form of bi-gram, trigram and so on (Appendix K). The first iteration showed to be unable to outperform the aforementioned LSTM. Looking closely into Tensorboad, both the weights for the last layer and convolutional layer revealed abnormal model behaviours (Figure 14). One reason that could explain this trend is linked to model size. In fact, if a network is too large for a certain dataset, channels can be relatively redundant. Thus, weights keep oscillating between residual values and consequently affect the validation accuracy and loss (Fig 15). The first solution tested was to lower the learning rate and add a Dropout before the output layer. This simple operation allowed to stabilise the model learning curve and increasing accuracy from 0.65 to 0.67 (Appendix L). Given that the model was still overfitting, it appeared that the last resort was to remove the fifth-gram component. While this operation returned the same precision and recall as the previous model, the ablation enabled to reach more homogenous precision across different classes. This trend can be illustrated in the figure below where the green/pink curves illustrate model performance prior ablation and the orange/blue ones after removing the last convolution.

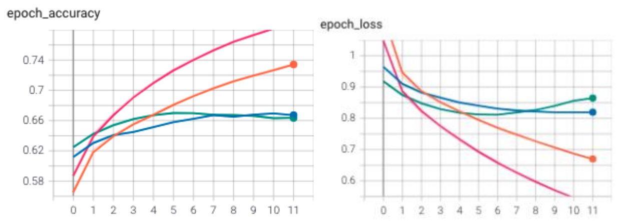


Fig. 15: *Learning curves for models before and after ablation*

Given the efficiency of the ablation study, further ablation of n-gram layers was also tested. However, this removal has degraded model performance. Thus, it was concluded that the best version possible of this architecture was reached.

5.5 Hybrid model: LSTM + CNN

The experiments involving the last model followed the same ablation approach mentioned previously. The initial model consisted of a simple LSTM layer followed by a series of convolutions and pooling layers (Appendix M). The initial version of the model performed relatively well and output an accuracy of 0.67 and a recall of 0.57. Just like in figure 14, the fourth gram layer was removed as part of the ablation study as it seemed to not contribute at all to the learning process.

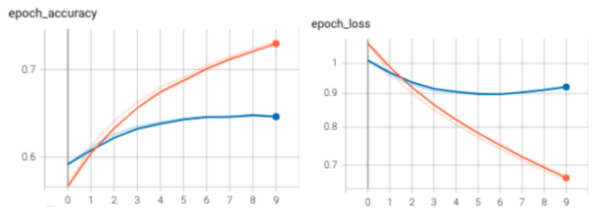


Fig. 16: *Learning curve for the hybrid model*

While the ablation showed exactly the same results, the learning curve of the simpler model showed to return a lower loss and slight less overfit (Appendix N). When looking at the learning curves below, it appeared that training loss started increasing at the 7th epoch, while accuracy kept increasing.

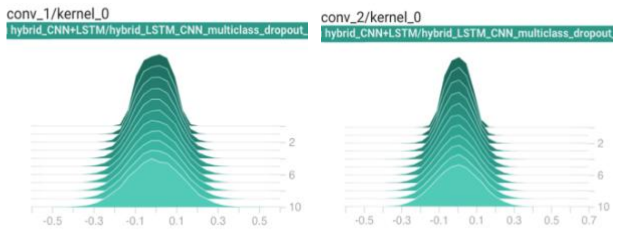


Fig. 17: *Weight distribution for the Convolution layers*

However, the weight histograms on Tensorboard did not reflect the same trend, as they showed to be clearly normally distributed (Fig17). In this kind of scenario, it could be argued that the model is overfitting and that its good generalisation is due to the possibility that the training and test splits are too similar. To understand this trend in a more granular level, the confusion matrix of model predictions was visualised at each epoch through TensorFlow callbacks. It could be observed that the model ability to correctly classify the extreme classes (very positive/very negative) decreased dramatically between the beginning of epoch 7 and the end of the training. In more detail, the probability of classifying correctly the class zero declined from 0.3 to 0.28. Concerning class 4, the probability of correctly classifying it remained unchanged. However, the one of predicting its neighbour class increased dramatically (From 0.55 to 0.59), which justifies the assumption that the shape of the learning curve was linked to overfitting.

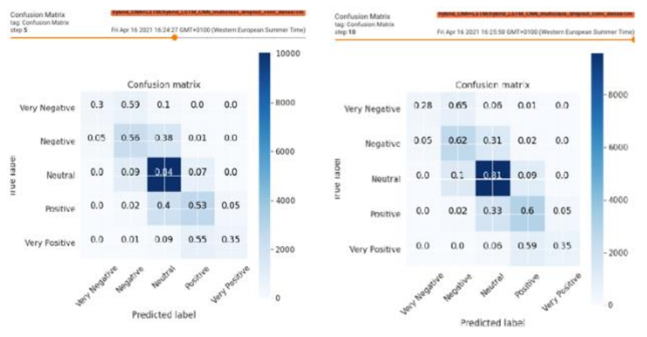


Fig. 18*: Model confusion matrix between epoch 5 and 10*

As a consequence, Dropout was added after the first and the last layers (Embedding and Dense). This simple operation allowed to obtain the best results so far across all models. Namely, the model achieved an accuracy of 0.68 with a macro-F-1 score of 0.59 (Appendix N). Given that these results were the best so far and obtained only with minimal effort, this model was selected as the "best performing model". While previous experiments were based on several empirical experiments, their aim was to identify the architecture that had the most potential. Therefore, following the logic outlined in section 4.5, the model was fine-tuned using data augmentation and a TensorFlow powered manually built grid search. The outcome of the parameter tuning was visualised on Tensor Board as shown in the figure below.

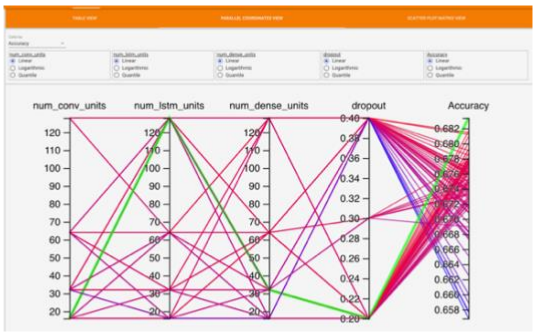


Fig. 19*: Tensorboard visualisation of hyper-parameter tuning*

The results obtained after tuning revealed to be dramatically higher than all tests undertaken so far. However, excessive regularisation led the model to suffer from underfitting. The, the aim of this exercise was to study the process of altering model architectures to increase performance, not necessarily to achieve state of the art results. Hence, the model was kept unchanged as it achieved a reasonable degree of performance.

5.6 State of the art Transformers: BERT

The final experiment involved solving the task using a state-of-the-art model. Due to computational limitations and its extremely long training time, it was not possible to undertake a detailed ablation study on this model. Thus, its initial version consisted of a succession of an embedding layer, a pretrained BERT layer and two dense layers before the output. The removal of the last dense layer showed to slightly improve the model. Despite the inability to manipulate model architecture, BERT returned the best results when compared to all other experiments. For instance, it allowed relatively homogenous predictions across classes. This model also returned an overall accuracy of 0.72 and a F1-macro of 0.63. Although the learning curve revealed strong overfitting, it is certain that the model can be further improved with more computation resource and time. Nevertheless, this last test has confirmed our initial hypothesis, being that even the simplest BERT model can outperform most Deep learning models.

	Precision	*Recall*	*F1*	*Test acc*	*Sub acc*
LR+TFIDF	0.19	0.20	0.15	0.49	0.434
LR+TFIDF+TSVD	0.50	0.31	0.33	0.55	0.501
LR+WORD2VEC	0.60	0.42	0.46	0.63	0.572
SIMPLE LSTM	0.60	0.55	0.57	0.67	0.619
MULTILAYER LSTM	0.62	0.56	0.58	0.68	0.627
CNN	0.62	0.51	0.55	0.67	0.608
CNN+LSTM	0.64	0.63	0.63	0.69	0.695
BERT	**0.66**	**0.61**	**0.63**	**0.72**	**0.712**

Fig. 20: *Final result table across all experiments*

6. DISCUSSION AND CONCLUSION

This report has described a wide range of model architectures in the context of movie review sentiment analysis. The study followed a gradual approach that distinguished the used of two principal paradigms. Namely, Supervised learning, Deep Neural networks. Supervised learning experiments allowed to prove the ability of semantic word embeddings (such as Word2Vec) to outperform traditional statistical approaches (Such as TFIDF). Experiments showed that despite heavy processing and transformations, baseline supervised models are not able to outperform the simplest LSTM model. The research could have been even further and test additional combinations of supervised models such as SVM or Random Forest with other embedding techniques (Glove, Fastetxt etc). However, this was not within the scope of this specific report and would have required additional time and computational resources. When moving to a deep learning approach, LSTM was used to compare different word embedding strategies. This specific model was chosen for its simplicity and ability to represent a reasonable baseline. The results showed that optimal results were achieved when using a pre-trained word2Vec model with no additional changes during training.
The extensive series of experiments allowed to get more familiar with a plethora of tools that allow to understand how to analytically improve model performance through visual tools such as Tensorboard. This software was particularly useful during the ablation study phase and also allowed to hyper-tune the best performing model, which ended up being a combination of CNN and LSTM architectures. Again, despite heavy processing and tuning, the best performing model was not able to outperform the BERT model, which justifies its status of State-of-the-art architecture in the NLP field. In conjunction to the ablation study, various regularisation techniques such as dropout or learning rate adjustment. These helped to adjust weigh histogram in general and improve model performance. However, improvements could have been brought to the followed methodology with more time and computing power. In fact, it would have been interesting to explore other approaches such as weight initialisation methods (such as Xavier or random normal). In addition, Finally, the choice of the best performing model was solely made on empirical experiments. A more systematic approach could have been to run the manual grid search for all models.

REFERENCES

[1] Tan, Q., Liu, N. and Hu, X., 2019. Deep Learning Representation for Social Network Analysis. Frontiers in Big Data, 2.

[2] Alabdullah, Bayan & Beloff, N. & White, Martin. (2018). Rise of Big Data – Issues and Challenges. 1-6. 10.1109/NCG.2018.8593166.

[3] Géron, A., 2020. Hands-On Machine Learning With Scikit-Learn, Keras, And Tensorflow. Beijing: O'Reilly.

[4] Vanderplas, J., n.d. Python Data Science Handbook.

[5] Khan, Taimoor & Durrani, Mehr & Ali, Armughan & Inayat, Irum & Khalid, Shehzad & Khan, Kamran. (2016). Sentiment analysis and the complex natural language. Complex Adaptive Systems Modeling. 4. 10.1186/s40294-016-0016-9.

[6] Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti A Zadeh. Feature extraction: foundations and applications, volume 207. Springer, 2008.

[7] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," Journal of machine learning research, vol. 3, no. Feb, pp. 1137–1155, 2003.

[8] V M, N. and Kumar R, D., 2019. Implementation on Text Classification Using Bag of Words Model. SSRN Electronic Journal,

[9] Bengfort, B., Bilbro, R. and Ojeda, T., n.d. Applied text analysis with Python ; enabling language-aware data pruducts with machine learning.

[10] Taboada, Maite & Brooke, Julian & Tofiloski, Milan & Voll, Kimberly & Stede, Manfred.(2011). Lexicon-Based Methods for Sentiment Analysis. Computational Linguistics. 37. 267-307.

[11] Jurafsky, D. and Martin, J., 2020. Speech and language processing. Uttar Pradesh (India): Pearson.

[12] Zou, Hui & Hastie, Trevor & Tibshirani, Robert. (2004). Sparse Principal Component Analysis. Policy. 1-30. 10.1198/106186006X113430.

[13] Drikvandi, Reza & Lawal, Olamide. (2020). Sparse Principal Component Analysis for Natural Language Processing. Annals of Data Science. Page: 1-17.

[14] Mikolov, Tomas & Sutskever, Ilya & Chen, Kai & Corrado, G.s & Dean, Jeffrey. (2013). Distributed Representations of Words and Phrases and their Compositionality. Advances in Neural Information Processing Systems. 26.

[15] Young, T., Hazarika, D., Poria, S. and Cambria, E., 2018. Recent Trends in Deep Learning Based Natural Language Processing *.IEEE Computational Intelligence Magazine*, 13(3), pp.55-75.

[16] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[17] Wu, Y., 2011. Data Augmentation, Internal Representation, and Unsupervised Learning. Journal of Computational and Graphical Statistics, 20(3), pp.581-583.

[18] Liu, Pengfei & Qiu, Xipeng & Huang, Xuanjing. (2016). Recurrent Neural Network for Text Classification with Multi-Task Learning.

[19] Tixier, Antoine. (2018). Notes on Deep Learning for NLP.

[20] A.Vaswani, N.Parmar, L.Jones, L.Kaiser, Gomez N.Shazeer , Attention is all you Need, 2019.

[21] Scikit-learn.org. 2021. *feature_extraction.text.Tfid -documentation*. [online] Available at:https://scikit learn.org /stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html [Accessed 17 April 2021].

[22] Munikar, Manish & Shakya, Sushil & Shrestha, Aakash. (2019). Fine-grained Sentiment Classification using BERT. 10.1109/AITB48515.2019.8947435.

[23] González, J., Hurtado, L. and Pla, F., 2021. TWilBert: Pre-trained deep bidirectional transformers for Spanish Twitter. Neurocomputing, 426, pp.58-69.

[24] Meyes, Richard & Lu, Melanie & Waubert de Puiseau, Constantin & Meisen, Tobias. (2019). Ablation Studies in Artificial Neural Networks.

[25] Wang, Yun & Metze, Florian. (2018). A comparison of five multiple instance learning pooling functions for sound event detection with weak labeling.

[26] Colón-Ruiz, C. and Segura-Bedmar, I., 2020. Comparing deep learning architectures for sentiment analysis on drug reviews. Journal of Biomedical Informatics, 110, p.103539.

[27] Suzuki, T., 2018. Fast learning rate of non-sparse multiple kernel learning and optimal regularization strategies. Electronic Journal of Statistics, 12(2).

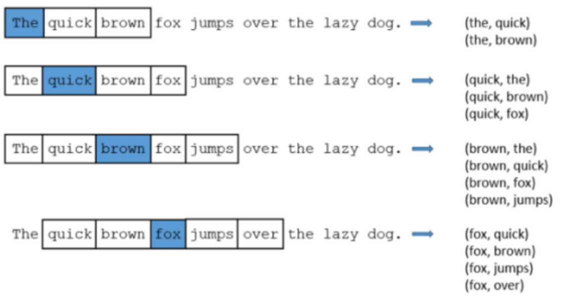
[28] B. Shu, F. Ren and Y. Bao, "Investigating Lstm with k-Max Pooling for Text Classification," 2018 11th International Conference on Intelligent Computation Technology (ICICTA10.1109/ICICTA.2018.00015.

[29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," The Journal of Machine Learning Research, vol. 15, no. 1, pp. 1929–1958, 2014.

[30] Scikit-learn.org. 2021. *sklearn.metrics.f1_score documentation*. [online] Available at: <https://scikit-learn.org /stable/modules/generated/sklearn.metrics.f1_score.html> [Accessed 17 April 2021].

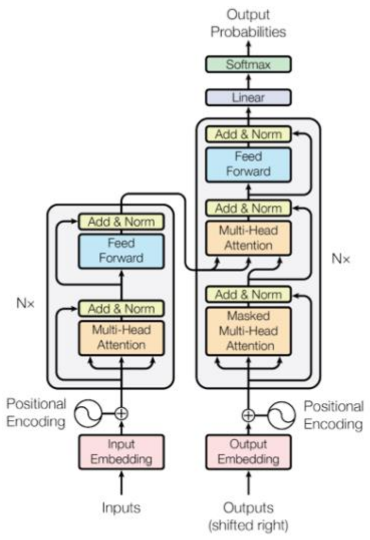
APPENDIX A: Negative sampling

The image below describes the mechanism employed by Word2Vec models to achieve negative sampling. This method allows the model to discard the importance of recurrent words within a sentence, while also decreasing the number of trainable parameters.


The quick brown fox jumps over the lazy dog.
(the, quick)
(the, brown)
The quick brown fox jumps over the lazy dog.
(quick, the)
(quick, brown)
(quick, fox)
The quick brown fox jumps over the lazy dog.
(brown, the)
(brown, quick)
(brown, fox)
(brown, jumps)
The quick brown fox jumps over the lazy dog.
(fox, quick)
(fox, brown)
(fox, jumps)
(fox, over)

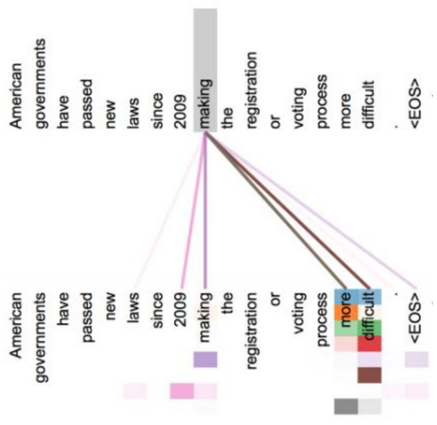
APPENDIX B: Attention, Transformer architecture

The figure below represents the encoder-decoder architecture of transformer models. While this architecture is mostly suitable for sequence processing such as text translation, other models as BERT leverage the same stacked encoder to solve text classification tasks.



APPENDIX C: Attention visualization in layer 5

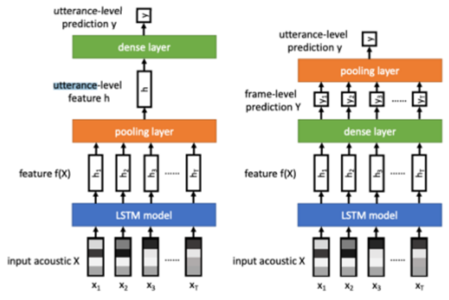
As illustrated in Appendix B, the transformer encoder stack leverages multiheaded attention to capture the meaning of words and their context within sentences. To do so, the model



establishes a representation of words and their association with all other words in a sequence. This process enables BERT to solve the inability of LSTM's to learn long term dependencies in extremely long inputs.

APPENDIX D: LSTM pooling strategies

The figure below illustrates two pooling strategies which were used to build custom models for experiments. The left plot illustrates the pooling on feature which consists of adding a pooling layer after the LSTM output.



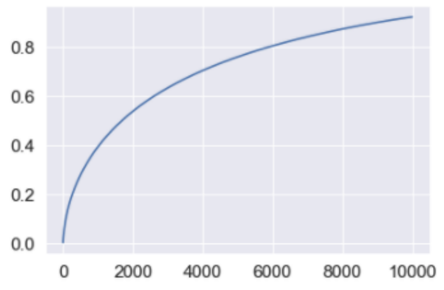
(a) Pooling on feature (b) Pooling on prediction

The second approach on the right consists of pooling after the LSTM outputs and states are passed to a dense layer.

APPENDIX E: Cumulative variance for TSVD

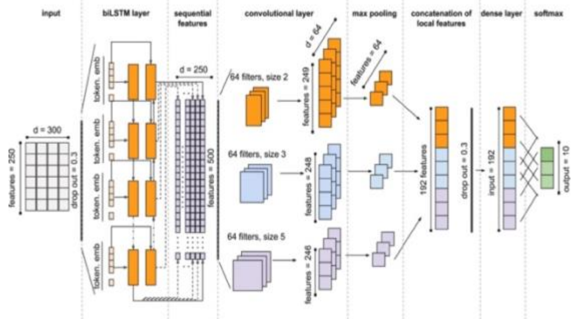
The graph represents the cumulative variance curve obtained after applying TSVD. As stated in the reports sections the number of components n was established so that it preserves a variance that is close to 1 after dimensionality reduction.

```
Initial train matrix shape is:  (109241, 16056)
Initial test shape is:  (23410, 16056)
PCA transformed train shape is:  (109241, 10000)
PCA transformed test shape is:  (23410, 10000)
```



APPENDIX F: LSTM+ CNN model architecture

This architecture leverages both the benefits of LSTM's and CNN's through a hybrid model. The motivation behind this



approach was to extract local features from the LSTM layer output, which corresponds to the sequence of hidden states return. Convolutional layers are then used to extract local information and aggregate the most important features using Global-Max-Pooling. The resulting operation is finally concatenated and passed to the final output SoftMax layer.

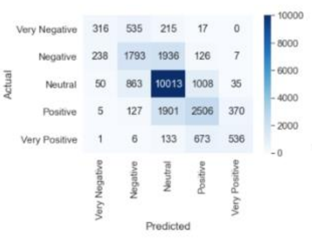
APPENDIX G: Single layer LSTM

This image depicts the simplest LSTM model that was tested in order to determine the best word embedding strategy. It simply consisted of an LSTM layer followed by a final output dense layer.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 52, 300)           5785500
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               219648
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 645
=================================================================
Total params: 6,005,793
Trainable params: 220,293
Non-trainable params: 5,785,500
```

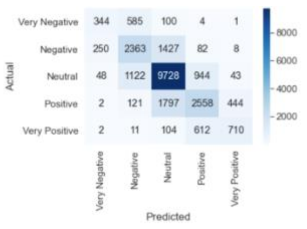
APPENDIX H: LSTM – Trained Word2Vec

This section summarises the different derived from the first experiments. The figure below shows model performance when used with a manually trained Word2Vec model.



```
              precision    recall  f1-score   support

           0       0.52      0.29      0.37      1083
           1       0.54      0.44      0.48      4100
           2       0.71      0.84      0.77     11969
           3       0.58      0.51      0.54      4909
           4       0.57      0.40      0.47      1349

    accuracy                           0.65     23410
   macro avg       0.58      0.49      0.53     23410
weighted avg       0.63      0.65      0.63     23410
```

The results below show that the pre-trained model is able to capture better the general meaning of words. Thus, it returned not only a better accuracy but also a greater precision and recall.



```
              precision    recall  f1-score   support

           0       0.53      0.33      0.41      1034
           1       0.56      0.57      0.57      4130
           2       0.74      0.82      0.78     11885
           3       0.61      0.52      0.56      4922
           4       0.59      0.49      0.54      1439

    accuracy                           0.67     23410
   macro avg       0.61      0.55      0.57     23410
weighted avg       0.66      0.67      0.66     23410
```

APPENDIX I: Multilayer LSTM model summary

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 52, 300)           5785500
_________________________________________________________________
bidirectional_25 (Bidirectio (None, 52, 256)           439296
_________________________________________________________________
dense_41 (Dense)             (None, 52, 64)            16448
_________________________________________________________________
bidirectional_26 (Bidirectio (None, 52, 256)           197632
_________________________________________________________________
dense_42 (Dense)             (None, 52, 64)            16448
_________________________________________________________________
bidirectional_27 (Bidirectio (None, 52, 256)           197632
_________________________________________________________________
dense_43 (Dense)             (None, 52, 32)            8224
_________________________________________________________________
global_max_pooling1d_11 (Glo (None, 32)                0
_________________________________________________________________
dense_44 (Dense)             (None, 32)                1056
_________________________________________________________________
dense_45 (Dense)             (None, 5)                 165
=================================================================
Total params: 6,662,401
Trainable params: 876,901
Non-trainable params: 5,785,500
```

APPENDIX J: Dense layer removal and dropout

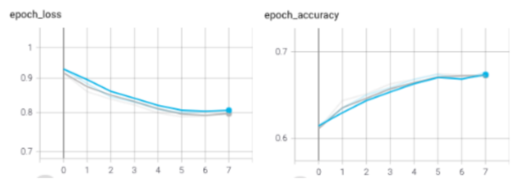
The model presented in Appendix I showed to provide enhanced results. However, it also showed to have reduced the overall F1-score due to apparent overfitting (Table below)

```
              precision    recall  f1-score   support

           0       0.61      0.31      0.41      1096
           1       0.57      0.52      0.54      4026
           2       0.74      0.85      0.79     12104
           3       0.60      0.58      0.59      4827
           4       0.66      0.33      0.44      1357

    accuracy                           0.68     23410
   macro avg       0.64      0.52      0.56     23410
weighted avg       0.67      0.68      0.67     23410
```

Thus, the last dense layer before output was removed after looking at its weight histograms. In addition, dropout was added to ensure that the model would truly stop overfitting.

```
           0       0.58      0.36      0.45      1061
           1       0.58      0.56      0.57      3946
           2       0.75      0.82      0.78     12052
           3       0.61      0.58      0.59      5013
           4       0.58      0.46      0.51      1338

    accuracy                           0.68     23410
   macro avg       0.62      0.56      0.58     23410
weighted avg       0.67      0.68      0.68     23410
```

While the table below shows the benefits of the ablation and dropout, we can also see that its learning curve (Grey) also displayed a lower loss.



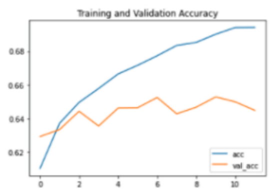
APPENDIX K: Initial parallel CNN model

The image below represents the very initial model at the start of the ablation study. As outlined in the report, the conv1d_3 as well as its corresponding dense layer was removed in order to improve the predictive power of the model.

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 52)]         0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 52, 300)      5785500     input_1[0][0]
__________________________________________________________________________________________________
conv1d (Conv1D)                 (None, 51, 100)      60100       embedding[0][0]
__________________________________________________________________________________________________
conv1d_1 (Conv1D)               (None, 50, 100)      90100       embedding[0][0]
__________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 49, 100)      120100      embedding[0][0]
__________________________________________________________________________________________________
conv1d_3 (Conv1D)               (None, 48, 100)      150100      embedding[0][0]
__________________________________________________________________________________________________
global_max_pooling1d (GlobalMax (None, 100)          0           conv1d[0][0]
__________________________________________________________________________________________________
global_max_pooling1d_1 (GlobalM (None, 100)          0           conv1d_1[0][0]
__________________________________________________________________________________________________
global_max_pooling1d_2 (GlobalM (None, 100)          0           conv1d_2[0][0]
__________________________________________________________________________________________________
global_max_pooling1d_3 (GlobalM (None, 100)          0           conv1d_3[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 128)          12928       global_max_pooling1d[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 128)          12928       global_max_pooling1d_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 128)          12928       global_max_pooling1d_2[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 128)          12928       global_max_pooling1d_3[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 512)          0           dense[0][0]
                                                                 dense_1[0][0]
                                                                 dense_2[0][0]
                                                                 dense_3[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 5)            2565        concatenate[0][0]
__________________________________________________________________________________________________
activation (Activation)         (None, 5)            0           dense_4[0][0]
==================================================================================================
Total params: 6,260,177
Trainable params: 474,677
Non-trainable params: 5,785,500
__________________________________________________________________________________________________
```

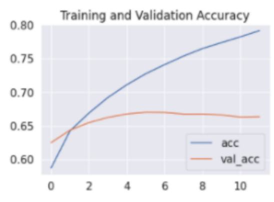
APPENDIX L: Model performance between adjusted learning rate and higher learning rate model

Initial model:



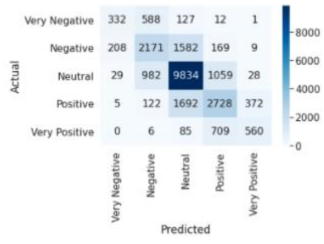
```
              precision    recall  f1-score   support

           0       0.46      0.44      0.45      1060
           1       0.56      0.42      0.48      4139
           2       0.72      0.81      0.76     11932
           3       0.55      0.60      0.57      4919
           4       0.61      0.26      0.36      1360

    accuracy                           0.65     23410
   macro avg       0.58      0.50      0.52     23410
weighted avg       0.64      0.65      0.63     23410
```

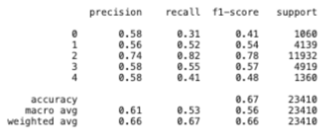
Adjusted Learning rate:



```
              precision    recall  f1-score   support

           0       0.61      0.23      0.34      1114
           1       0.55      0.56      0.55      4160
           2       0.73      0.83      0.78     11781
           3       0.61      0.54      0.58      4990
           4       0.60      0.40      0.48      1365

    accuracy                           0.67     23410
   macro avg       0.62      0.51      0.55     23410
weighted avg       0.66      0.67      0.66     23410
```

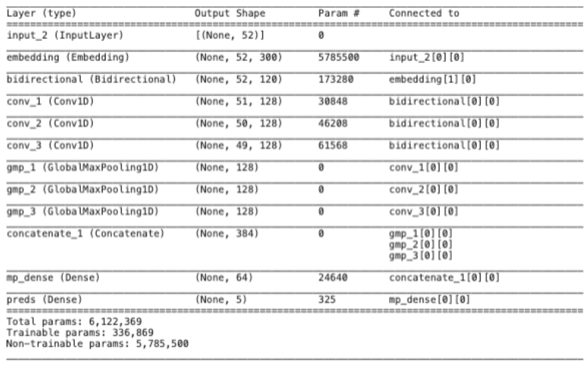
In addition to adjusting the learning rate, the ablation the component described in the report also showed to allow similar performance with less complexity.




```
              precision    recall  f1-score   support

           0       0.58      0.31      0.41      1060
           1       0.56      0.52      0.54      4139
           2       0.74      0.82      0.78     11932
           3       0.58      0.55      0.57      4919
           4       0.58      0.41      0.48      1360

    accuracy                           0.67     23410
   macro avg       0.61      0.53      0.56     23410
weighted avg       0.66      0.67      0.66     23410
```

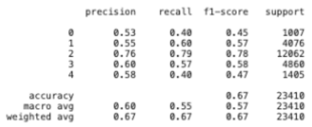
APPENDIX M:

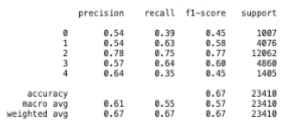
Following a similar approach used for the simple CNN model, the model below represented an initial version where the third Convolutional layer was removed. As described in the report this model architecture allowed to get better or equivalent results compared to other models with very minimal effort. Thus, it was selected as the 'best performing model' and was fine-tuned using a manual grid search and text augmentation.


```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            [(None, 52)]         0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 52, 300)      5785500     input_2[0][0]
__________________________________________________________________________________________________
bidirectional (Bidirectional)   (None, 52, 120)      173280      embedding[1][0]
__________________________________________________________________________________________________
conv_1 (Conv1D)                 (None, 51, 128)      30848       bidirectional[0][0]
__________________________________________________________________________________________________
conv_2 (Conv1D)                 (None, 50, 128)      46208       bidirectional[0][0]
__________________________________________________________________________________________________
conv_3 (Conv1D)                 (None, 49, 128)      61568       bidirectional[0][0]
__________________________________________________________________________________________________
gmp_1 (GlobalMaxPooling1D)      (None, 128)          0           conv_1[0][0]
__________________________________________________________________________________________________
gmp_2 (GlobalMaxPooling1D)      (None, 128)          0           conv_2[0][0]
__________________________________________________________________________________________________
gmp_3 (GlobalMaxPooling1D)      (None, 128)          0           conv_3[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 384)          0           gmp_1[0][0]
                                                                 gmp_2[0][0]
                                                                 gmp_3[0][0]
__________________________________________________________________________________________________
mp_dense (Dense)                (None, 64)           24640       concatenate_1[0][0]
__________________________________________________________________________________________________
preds (Dense)                   (None, 5)            325         mp_dense[0][0]
==================================================================================================
Total params: 6,122,369
Trainable params: 336,869
Non-trainable params: 5,785,500
__________________________________________________________________________________________________
```

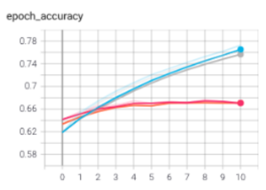
APPENDIX N:

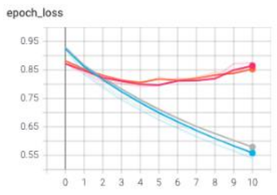
The tables below illustrate the model performance before and after the removal of the third convolution layer. While this change had no effect on performance metrics, it has allowed to obtain more a homogenous precision across classes.


```
              precision    recall  f1-score   support

           0       0.53      0.40      0.45      1007
           1       0.55      0.60      0.57      4076
           2       0.76      0.79      0.78     12062
           3       0.60      0.57      0.58      4860
           4       0.58      0.40      0.47      1405

    accuracy                           0.67     23410
   macro avg       0.60      0.55      0.57     23410
weighted avg       0.67      0.67      0.67     23410
```


```
              precision    recall  f1-score   support

           0       0.54      0.39      0.45      1007
           1       0.54      0.63      0.58      4076
           2       0.78      0.75      0.77     12062
           3       0.57      0.64      0.60      4860
           4       0.64      0.35      0.45      1405

    accuracy                           0.67     23410
   macro avg       0.61      0.55      0.57     23410
weighted avg       0.67      0.67      0.67     23410
```

This trend was also reflected in the learning curves where the model after ablation (Grey and orange lines).





Despite the result improvements the weight distribution of the remaining components showed to be gaussian. Consequently, it was not possible to make to further ablations unless through random experiments. Hence, overfitting was fought by adding dropout after the embedding and dense layers. This change allowed to attain the best results so far.

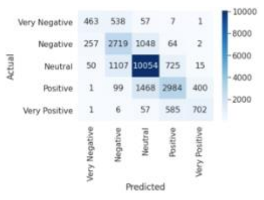
```
              precision    recall  f1-score   support

           0       0.51      0.47      0.49      1007
           1       0.56      0.58      0.57      4076
           2       0.76      0.80      0.78     12062
           3       0.61      0.56      0.58      4860
           4       0.61      0.44      0.51      1405

    accuracy                           0.68     23410
   macro avg       0.61      0.57      0.59     23410
weighted avg       0.67      0.68      0.67     23410
```

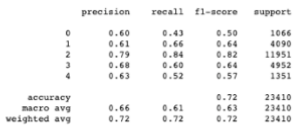
After hyper tuning the model. It returned significantly higher results. As shown below.

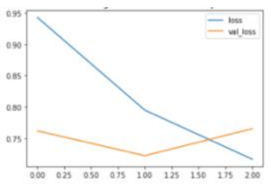
```
              precision    recall  f1-score   support

           0       0.58      0.53      0.55      1114
           1       0.55      0.71      0.62      4160
           2       0.83      0.71      0.77     11781
           3       0.60      0.71      0.65      4990
           4       0.64      0.51      0.57      1365

    accuracy                           0.69     23410
   macro avg       0.64      0.63      0.63     23410
weighted avg       0.71      0.69      0.69     23410
```

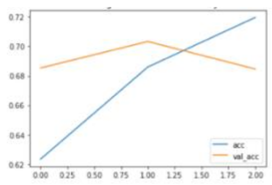
APPENDIX O: BERT results

The chart below illustrates the results obtained using BERT.




```
              precision    recall  f1-score   support

           0       0.60      0.43      0.50      1066
           1       0.61      0.66      0.64      4090
           2       0.79      0.84      0.82     11951
           3       0.68      0.60      0.64      4952
           4       0.63      0.52      0.57      1351

    accuracy                           0.72     23410
   macro avg       0.66      0.61      0.63     23410
weighted avg       0.72      0.72      0.72     23410
```





Link to the GITHUB REPO :
https://github.com/zceihzi/AMLSII_final_assignment