UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

Twitter Sentiment Analysis and Tweet Extraction

Deep Learning Project Report

Rohith Teja MITTAKOLA
Yogesh Kumar PILLI

Abstract

In the social media, Twitter has been the most engaging platform for a long time and many companies, political personalities, celebrities have their presence in twitter which makes it a great place for having discussions by millions of users about various topics ranging from laws passed by the parliament to the new movie releases. This makes it interesting to analyse the sentiments of the tweets to see if a tweet by a certain user has positive or negative impact on the community. Going a step further, we can also find which words of the tweet contribute to the sentiment. The analysis of public reaction can be easily done using the sentiment analysis and the keyword extraction of the tweets.

In this report, we present the sentiment analysis of tweets using various deep learning algorithms and compare their performance using different metrics. Also provide a method to perform keyword extraction of tweets. The dataset [6] that we considered encompasses a broad set of tweets which are classified into 3 different types of sentiments, namely, positive, negative and neutral.

Keywords: Deep Learning, Neural Networks, Sentiment Analysis, Tweet Extraction

January 2021

Contents

List of Figures

List of Tables

1 Introduction

Everyday, on an average, 500 million tweets are being sent on the social networking site, Twitter, relating to a myriad number of topics. Because of this, a huge dataset of tweets can be generated on a daily basis which has the valuable information relating to the public opinion on different topics. Using methods in natural language processing, we can break down the tweets into different public sentiment like positive, neutral and negative. We can also construct more detailed sentiments like strongly positive, weakly positive etc which captures the sentiment in fine details. The analysis will explain the number of people who are reacting to a topic positively or negatively and this can help understand why the public opinion is inclined in such a manner. With such analysis, it will enable the companies to fix their products to better suit the majority of the public, help politicians take correct stance on a sensitive topic and so on.

In other words, we can describe the sentiment analysis as a part of computational linguistics which focuses on finding the polarity of an opinion/comment or in this case, a tweet. It can capture different emotions exhibited by the person by analysing the structure of the words. There are various tools present in the Natural Language Processing which enables us to input the text in "readable" format to various algorithms. This allows us to use the text in various machine learning tasks. The sentiment analysis is an example of text classification and the tweet extraction is same as keyword extraction.

In this report, we study various machine learning algorithms, especially the ones involving neural network architectures to perform tasks like text classification and keyword extraction on the tweets. We first explain the overview of neural network methods that we are planning to experiment and then some natural language processing tools which are important in the context of the tasks mentioned above. Then, we build 3 cases to show different types of experiments we performed, out of which 2 are text classification methods on different text inputs and 1 keyword extraction task. The objective of building different cases is to analyse how the results are changing depending on the input and also to establish a relationship between different techniques.

Some interesting results were obtained during our experimentation, where the sentiment analysis gives better results if we use the output from the tweet extraction. In other words, the text extracted from the tweets was more suitable for the sentiment analysis rather than the text from the whole tweet. With our experiments, we try to find the reason behind that and also provide some insights in selecting the best algorithm to use for sentiment analysis task.

2 Overview of Neural Network Methods

The complex architecture of the neural networks enables us to evaluate the relationship between the input text and the output analysis. We have different architectures for the neural networks which can be employed for a certain task but we are focusing on the popular architectures (which are listed below) for our analysis. Neural network architectures can be used for solving various problems in machine learning and in this report we focus on the case of multi-classification problem.

2.1 Components of a neural network

In this subsection, we discuss the various components involved in building a neural networks and some concepts which are helpful to understand the architecture. The artificial neural networks are composed of several nodes called neurons which behave similar to the neurons in the human brain. We have several layers of neurons which are connected to each other to make a network. The first layer is called the input layer and the last layer is called the output layer. The layers between the input and output layers are called hidden layers.

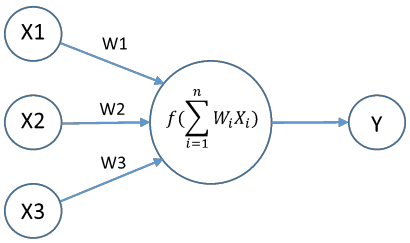


Figure 1: Neuron illustration

Figure 1 shows how a single neuron (in the hidden layer) works in the network. The neuron has input from various neurons in the input layer with weights attached to those values, which are summed up before applying an activation function. Then, the output of this neuron becomes the input to the next layer. The output of a neuron is determined by the activation function.

There are different activation functions like binary step function (which activates the neuron only if the value is above a certain threshold), linear activation function (the output is proportional to the input) and non-linear activation functions. Non-linear activation functions are widely used in the neural network architectures. Some of them are listed below:

1. Sigmoid
2. Tanh
3. ReLU (Rectified Linear Unit)
4. Selu
5. Leaky ReLU
6. Softmax
7. elu (Exponential Linear Unit)

A loss function is used to measure the difference between the model prediction and the actual value. Our aim is to decrease the loss as much as possible in order to get better predictions that are close to the actual value. During the training, the neural network tries to find the weights which reduce the loss function as a whole. The weights are modified by using optimization technique to decide by how much value a weight should be changed. The weights are calculated until we run out of the epochs or the algorithm converges to a minimum loss. There are several optimization techniques that can be used with neutral networks, some of them are shown below:

1. Stochastic Gradient Descent (SGD)

2. Adam
3. RMSprop
4. AdaDelta

2.2 Multilayer Perceptron (MLP)

This is a very basic neural network architecture which consists of multiple hidden layers and it is a type of feed forward network where all layers are fully connected.

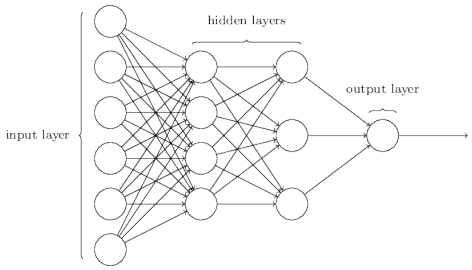


Figure 2: Multilayer perceptron

Source: [5]

As shown in figure 2, MLP consists of an input layer where the data is fed into the neural network. We can have a series of hidden layers with each layer having a set of neurons with an activation function. The output layer shows the interpretation of the model with the input data. If it is a case of a multi label problem, then the output neurons should be same as the number of unique labels.

2.3 Convolutional neural network (CNN)

Initially convolutional neural networks are developed for image processing and image classification but they can be employed on text classification tasks also. The network uses convolution and pooling as two main operations which function as feature extractors for the model. The output from convolution and pooling is connected to a fully connected multilayer perceptron. The final output layer of CNN is similar to that of an MLP.

Convolution is a process by which we select a filter or a kernel of a predefined size and move this filter along the text values (the text in the sentence are converted into number using an embedding technique which is explained in the following sections) and multiply the values corresponding to the filter values. Then, all the values in the filter are added to make it as the first feature in the result array. The filter is moved along the text values depending on the stride length (it determines how many steps the filter can take everytime it moves). The result array obtained is called as the convoluted feature. Every time the filter is moving along the text values, it is picking up the ngrams (as shown in figure 4) and size of the filter and ngrams obtained is important for the text classification output. [11].
Pooling is a technique used to downsample the convoluted features. We use a similar filter but the operation performed using the filter is different than that of convolution. Different operations can be performed like max-pooling (picking the max value from the filter), average-pooling etc depending

on the task. The main purpose of the pooling technique is to reduce the size of the convoluted feature. [3] Padding is used if we want to keep the size of the array same as the input, then "same padding" makes sure that the size of the array after convolution process is the same.

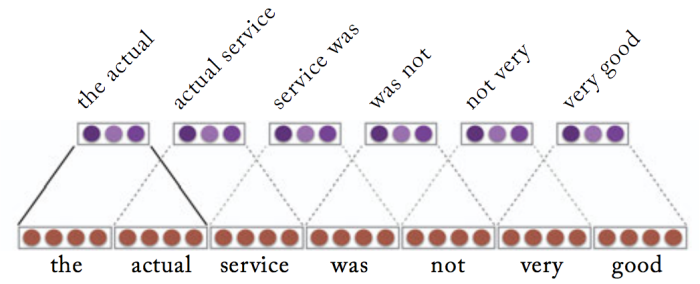


Figure 3: 1D convolution for text

Source: [3]

Figure 4 shows the convolution process for the sentence in 1 dimension. Here, the filter is chosen with the size 2 so the words captured are of length 2 (in the top layer). The input text is represented as a vector embedding of 4 dimensions and after the convolution, the dimension is reduced to 3.

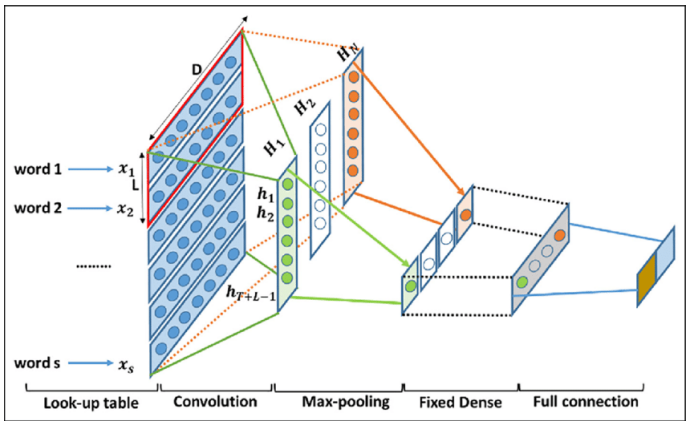


Figure 4: CNN for text classification

Source: [13]

2.4 Recurrent neural network (RNN)

Recurrent neural networks are recurrent for a certain number of timesteps and work in a way such that the input from the previous time step is also added to the current input.

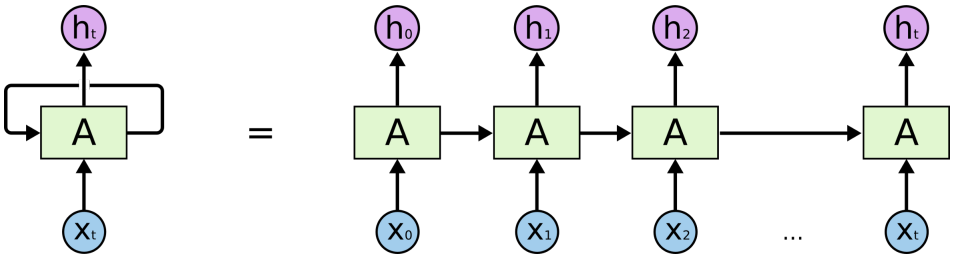


Figure 5: RNN illustration

Source: [7]

Figure 5 shows how the input is fed into the network in a loop. This way the network memorizes the sequence of the words in the sentence. This method is useful for preserving the sentence structure where the the context of the words is taken into account while training. During back propagation through time, we face the problem of vanishing gradient where the gradient becomes smaller and smaller which finally has very less effect in modifying the weights. This problem is fixed with the LSTM network.

2.5 Long short-term memory (LSTM)

Standard RNNs were unable to learn long-term dependencies as the gap between two time steps becomes large. To address this issue, LSTM was first introduced in [10] and reemerged as a successful architecture since Ilya et al [9] obtained remarkable performance in statistical machine translation. Many researchers have done lot of work in publishing the different variants of LSTM which were proposed, we have adopted the standard architecture [10] from this renowned work.

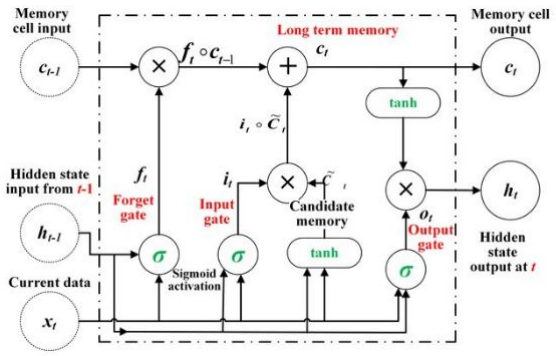


Figure 6: LSTM cell illustration

The LSTM architecture has a range of repeated modules for each time step as in a standard RNN. In Figure 6 at each time step, the output of the module is controlled by a set of gates in Rd as a function of the old hidden state h_{t1} and the input at the current time step x_t: the forget gate f_t, the input gate i_t, and the output gate o_t. These gates collectively decide how to update the current memory cell c_t and the current hidden state h_t.

We used to denote the memory dimension in the LSTM and all vectors in this architecture share the same dimension. The LSTM transition functions are defined as follows:

- **Forget gate**: This gate is known as sigmoid layer gate which takes the output at t-1 intervals and the current input at time t and then makes them together into a single tensor. After this it applies linear transformation following by a sigmoid layer. The output of this gate which varies inbetween 0 and 1 due to the sigmoid layer. Resulting number is then multiplied with the internal state of the network, and that is why the gate is called forget gate. If ft =0 ,then the previous internal state of the network is completely forgotten, while if ft =1, it will be passed.

$$f_t = \sigma(W_f\ [h_{t1}, x_t] + b_f)$$

- **Input gate**: This gate takes the preceding output together with the new inputs and allows them via same another sigmoid layer.This state results a value which is in-between 0 and 1. Then result of this input gate is combined with the outpur of the candidate layer.

$$i_t = \sigma(W_i \cdot [h_{t1}, x_t] + b_i)$$

- **Memory Cell**: This gate applies hyperbolic tangent to combine the previous and input gate output together,which results in candidate vector.This vector is then added to the internal state by the following rules:

$$q_t = \tanh(W_q \cdot [h_{t1}, x_t] + b_q)$$

$$c_t = ftc_{t1} + i_t q_t)$$

  The preceding state is multiplied by the forget gate, and then added up to fraction as new candidate which allowed by the output gate

- **Output Gate**: This gate controls how much of the internal state is passed to the output and works in a similar manner to the other gates.

$$o_t = Sigmoid(W_o \cdot [h_{t1}, x_t] + b_o)$$

$$h_t = o_t tanh(c_t)$$

2.6 Gated recurrent unit (GRU)

Unlike LSTM, GRU has no cell state and uses a hidden state for information transfer.

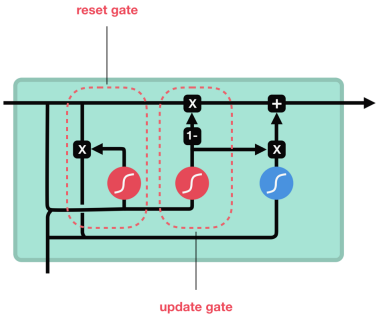


Figure 7: GRU cell illustration

Figure 9 shows the two gates, reset and update gates present in the cell. Update gate decides which information to be remembered and which to forget. Due to the lesser gates in GRU, it is usually faster in training times than LSTM. Advancement over LSTM, GRU also can be used to enhance the memory storage of recurrent neural network.Hidden layers can be used as settlement for vanishing gradient problem of RNN.

2.7 Bi-directional LSTM

This architecture uses a forward LSTM and a backward LSTM together to better understand the context of a sentence.

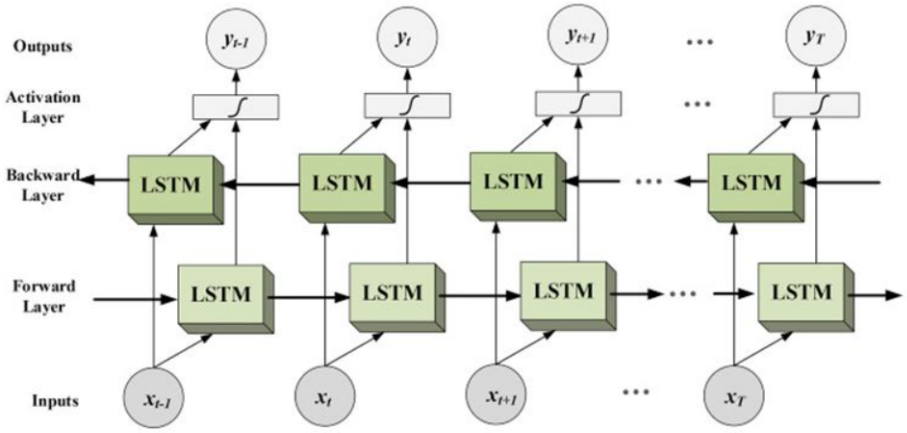


Figure 8: Bidirectional architecture

Source: [4]

The main thought process behind the Bidirectional LSTM is direct.which includes duplicating the first recurrent layer in the architecture then organizing this result into the result as the first layer and then giving the reversed result of the input sequence to the another layer.Bidirectional Recurrent Neural network can be trained upon all the future and past inputs for the specific timesteps. Divided the state of neurons in BiLSTM is done in forward state which indicates the positive time directions and vice versa for the backward state.

2.8 Transformers

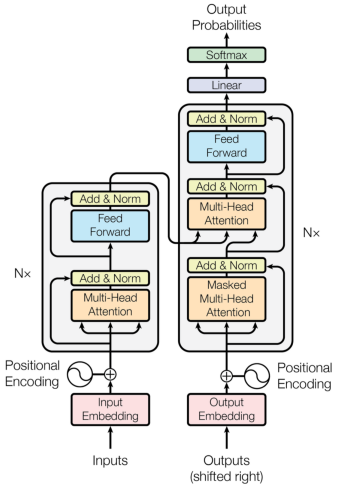


Figure 9: Transformer architecture

Source: [16]

The working of a transformer is explained as follows:

1. Firstly, the sentences are represented as an embedding where each word has its own latent representation.
2. Then a positional encoding is applied (added to the word embeddings). Positional encoders are used to store the information of the distance between words in the sentence. Now, we have the word embeddings with positional information.
3. An attention layer is built where attention vectors are generated. These vectors store the information of the sentence where every word is assigned an importance, so as to know which word in the sentence we should focus on. Then a feed forward network is connected to the attention layer, making the entire thing, a part of the encoder block. We have a decoder block, that takes the output embedding and add the positional encoding and attention layers similar to that of the encoding block. [16]
4. The both encoder and decoder blocks are connected with a feed forward network and a linear layer , followed by a softmax layer which predicts the probabilities of the classes.

Transformers take a long time to train due to the complex architecture but usually perform much better than other neural network methods. There exist several pretrained transformer models like BERT, XLNet, ROBERTa which are explored during our experiments. The code used in our experiments relating to transformer architecture is taken from [2]. A quick description of the transformer models is shown below:

1. BERT: Bidirectional Encoder Representations from Transformers, is a transformer based architecture developed by Google. We used a BERT base model which is a 12-layer, 768-hidden, 12-heads, 110 million parameter neural network architecture. [8]

2. XLNet: As descibed by the paper [17] , XLNet is a generalized autoregressive pretraining method which is supposed to provide a solution to the shortcomings of BERT.

3. ROBERTa: This is a transformer architecture built on top of BERT model, which is being researched by Facebook. [12]

3 Overview of Natural Language Processing methods

In this section, we explain the various techniques used in natural language processing which helps for tasks like text classification. Text classification is a task where our objective is to classify a given text into different classes.

3.1 Text Preprocessing

Text preprocessing is a task where we convert the text into simple format that is more convenient to process. It involves removing unwanted words and characters, modifying some words etc. Although we can use the text without preprocessing for the purpose of classification, but studies have shown that the algorithm perform bettwe with some forms of simple preprocessing like lower casing the text and tokenization. [15]. For our experimentation, we used the following preprocessing techniques:

1. Lower casing the entire text
2. Removing URLs and emails which are not helpful when doing text classification
3. Removing numbers
4. Removing white spaces
5. Removing punctuations and special characters

There are more advanced methods of preprocessing like replacing the contractions with full form (Ex. the word "couldn't" is replaced with "could not"). But we have not employed such methods in our experiments.

3.2 Text Vectorization

The input to the neural networks should be in numerical format. Hence, the text is converted into numerical representation and this process is called text vectorization. There are some methods which help us with text vectorization and they are discussed below:

1. Bag of Words:
   This is a simple and straightforward method for numerical representation of text. Firstly, we build a dictionary of all the words present in the given sentences and for each sentence we generate a vector (which is same size as the number of words in the dictionary) where if the words present in the sentence are denoted by 1 and 0 otherwise. This technique is not useful in the case of large text because of the high dimensionality of the vector generated. The empirical performance of the bag of words is explained in this paper [18].

2. Word embeddings:
   Word embedding is a method where each word in a sentence is mapped to a vector of d dimensions such that the words which have close semantic similarity are represented closer to each

other in the latent space. This is an advanced technique than the bag of words which carries more context of the sentence in a low dimensional space.

Embedding layer is a method by which we can find the word embeddings while the neural network is being trained. The text has to be input in one-hot vector format where a number is assigned to every word in the sentence (similar to bag of words).

Global Vectors for Word Representation, or GloVe [14] is a pretrained word embeddings which has been trained on 2 billion tweets with 27 billion tokens. For our experiments we used the GloVe embeddings of 100 dimensions. This is based on the Word2Vec method.

4 Experiments

4.1 Dataset

The dataset is taken from the kaggle competition "Tweet sentiment extraction" [6]. It consists of 27481 tweets which are denoted as "Full text" for the sake of our experiments. It also consists a column having text that strongly supports the sentiment of the tweet. This part is denoted as "Selected text" in our experiments. Finally, every tweet is labeled with a sentiment of being positive, negative or neutral. Our analysis is divided into 3 cases which are mentioned in the following subsections.

4.1.1 Exploratory Data Analysis

As explained earlier the dataset is labeled with different sentiments. In this section the distribution of the different sentiment classes is shown in the figure 10

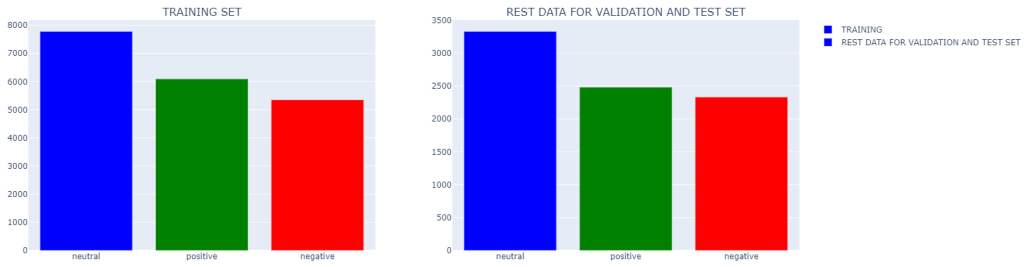


Figure 10: Sentiment Labels in Training and Rest of the Data

The dataset is split into train, validation and test sets. The train set is used in the training phase of the neural network (70% of the data is included in the train set and the rest 30% is split into validation and test set equally) and validation set is used to tune the hyperparameters. Finally, the test set is used to analyse the performance of the final model as shown in figure 11

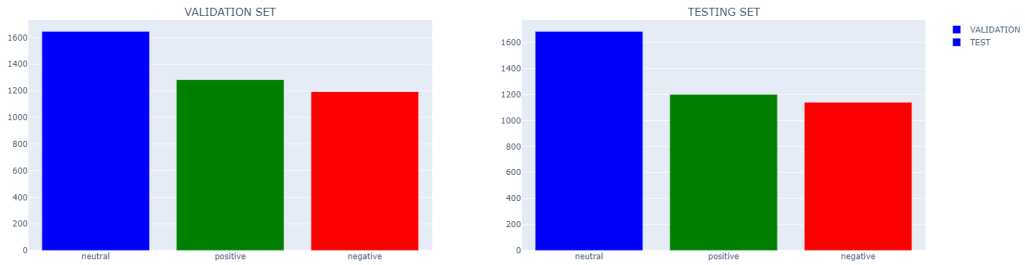


Figure 11: Sentiment Labels in Validation and Test Set

The performance metrics used are accuracy score and F1 scores. Accuracy scores are important when we are looking for true positives and true negatives while, F1 scores are important in the case of false positives and false negatives. Also, F1 scores become important in the case of imbalanced dataset. Our tweet dataset is slightly imbalanced but not much with neutral class is dominating over other classes in our dataset. We verified if this slight class imbalancy in the dataset affects the accuracy and F1 scores obtained during sentiment analysis.

For our experiments, it is also important to understand the distribution of total number of words present in every tweet in the entire dataset. The graph below shows the distribution for the number of words in the tweets separately for each sentiment: (figure 12)

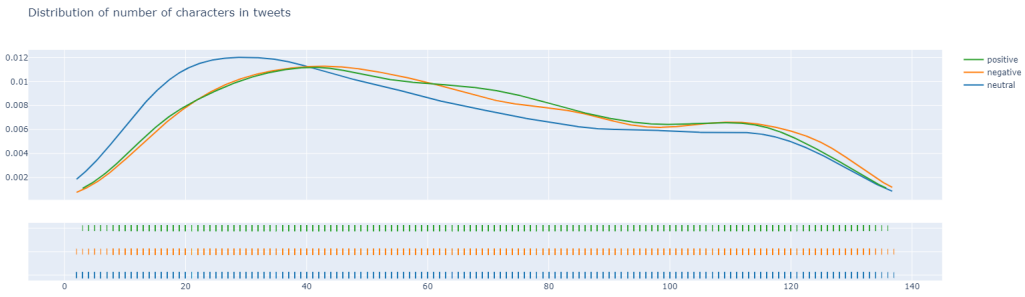


Figure 12: Distribution of number of Characters in tweets

We observed that the maximum tweet length is 32 and minimum is 1. We have taken this into account while performing the experiments.

Basically we are dealing with the textual data in for classification of the sentiment labels. We need to understand the most frequent words that help in text classification. This can be done in the form of word cloud representation. Word clouds are important to extract the top frequent words in a dataset. In the figure 13 we picked 3 random tweets (one for each sentiment) from the dataset and illustrated their word cloud representations.

Figure 13: Word Cloud Representation on FullText

Now, we perform the visualization for same tweets present in the selected text part of dataset to see if the frequent most are same in both the case or if there any change in the frequent words.

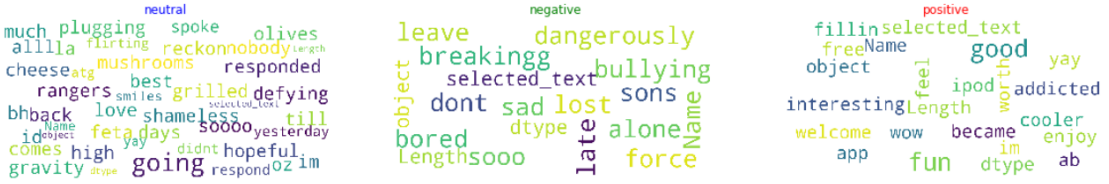


Figure 14: Word Cloud Representation on Selected Text

4.2 Experimental setup

Our experimental setup is split into 3 cases where the following neural network methods are build for Case 1 and Case 2. (The best performing model in case 1 and case 2 is used for modeling the Case 3):

1. Multilayer Perceptron (MLP)
2. Convolutional neural network (CNN)
3. Recurrent neural network (RNN, LSTM, GRU, Bi-directional LSTM)
4. Transformer architecture (BERT, XLNet, ROBERTa)

A total of 9 algorithms are checked with every case. The cases are explained in the following subsections.

4.3 Case 1: Sentiment Analysis using Full text

In this case, the sentiment analysis is performed on the Full text by using several neural network methods. The text vectorization is done in 2 different types namely,

- Pretrained Glove embeddings: GLoVe embedding trained on 2 billion tweets is used with a dimension of 100
- Embedding layer: The embeddings are generated while training the model.

We compare the two types of embedding techniques by analysing the accuracy and F1 scores.

4.3.1 Results

	Pretrained embs		Embedding layer		Time(seconds)
	Acc. score	F1 score	Acc. score	F1 score	
MLP	69.46	69.53	70.09	70.05	49.04
CNN	72.78	72.79	70.24	68.96	221.50
RNN	71.69	71.73	67.47	67.49	251.91
LSTM	74.63	73.65	72.13	71.04	160.97
GRU	72.08	72.13	71.50	71.44	133.69
Bi-LSTM	73.95	73.99	71.35	71.52	183.44

Table 1: Performance metrics for Case 1: Sentiment analysis with Full text

4.3.2 Results (with Transformers)

	Acc. score	F1 score	Time(minutes)
BERT	78.13	78.13	15.3
XLNet	77.98	77.96	15.50
ROBERTa	77.03	77.01	11.30

Table 2: Performance metrics for Case 1 with Transformers

4.4 Case 2: Sentiment Analysis using Selected text

The sentiment analysis performed in this case is different from Case 1. The input text used for this case is the Selected text. All the experiments performed are identical for both cases, so as to have a good comparison of the performance of neural network methods in both cases.

4.4.1 Results

	Pretrained embs		Embedding layer		Time(seconds)
	Acc. score	F1 score	Acc. score	F1 score	
MLP	83.07	83.45	82.17	82.99	24.22
CNN	84.88	84.92	81.97	81.28	96.62
RNN	82.51	82.43	81.27	81.31	221.97
LSTM	84.96	84.99	84.04	84.06	120.55
GRU	84.08	84.18	84.64	84.64	118.86
Bi-LSTM	85.25	85.27	83.07	83.06	134.78

Table 3: Performance metrics for Case 2: Sentiment analysis with Selected text

4.4.2 Results (with Transformers)

	Acc. score	F1 score	Time(minutes)
BERT	88.13	88.12	15.2
XLNet	83.87	84.91	15.12
ROBERTa	87.33	87.33	13

Table 4: Performance metrics for Case 2 with Transformers

4.5 Case 3: Tweet extraction (Full text to Selected text)

As we have seen the difference in performance of the neural network models with full text and also selected text, we test a method to convert the full text into selected text. For this purpose, we use the top performing models in the case 1 and case 2.

From the results shown in Case 1 and Case 2 (in the previous subsections), it is clear that BERT is the best performing model of all. So we use BERT architecture (BERT base uncased model) for tweet extraction. The process of tweet extraction is done as follows [1]:

1. Data is preprocessed by removing unwanted characters.
2. BERT tokenizer is used to split the sentences into separate words and the dataset is split into train, validation and test sets.
3. Start and end positions are captured in the train set by matching the positions of Full text and selected text. And for the test set these values are set to a random integer. (Don't include the examples with no start and end positions)
4. Now, the examples in train and test are separated with respect to the sentiment (positive, neutral or negative).
5. Two models with BERT architecture are built for the positive and negative examples. We do not consider the neutral examples for tweet extraction.
6. The models are trained for 3 epochs with 32 batch size and maximum sequence length set to 60.
7. For the test data, the start and end positions are predicted. We use Jaccard score as metric to evaluate the predicted text. Jaccard score tells us how similar two sentences are and higher the score, it means that the sentences are more similar.
8. From our experiment, we got a jaccard score of 0.7022 on the test dataset.

4.6 Model Comparison

In the previous sections, the results are shown in tabular format for each case. The figure 15 shows the comparison of accuracy scores for all the neural network methods in one place. Since the pretrained embedding layer is the overall best performer in text vectorization, we compare the accuracy scores obtained by using GloVe embeddings. Both Case 1 and Case 2 are represented in the figure. We can see that the transformer models (especially BERT) outperform the other neural network methods. This is explained in more detail in the following sections.

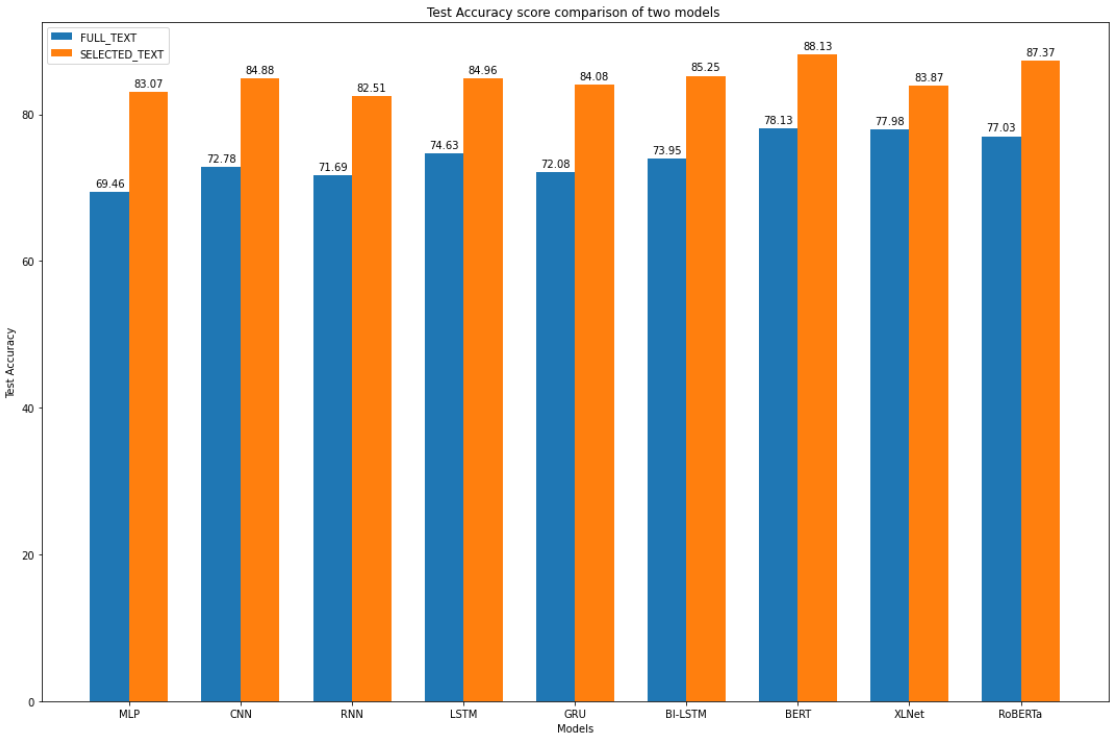


Figure 15: Accuracy measure comparison (with GloVe embeddings) for different models

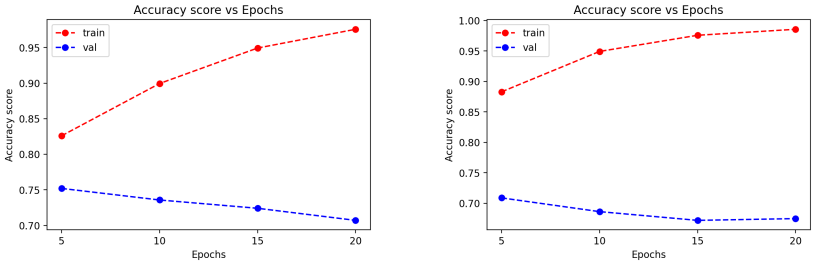
We have already mentioned why we are also considering the F1 measure in our performance metrics (to focus on data imbalancy and false negative as well as false positives in our results.) In Figure 25 we can see the same trends as in the Figure 15 were the F1 measure is highest for the BERT model.

5 Analysis of the experiments (of Case 1)

In this section, we analyse the performance of the neural network models in detail by checking the training and validation scores for each model. (Since, the number of neural network methods are large, we show the analysis only on Case 1, where the algorithms have relatively low performance when compared to case 2).

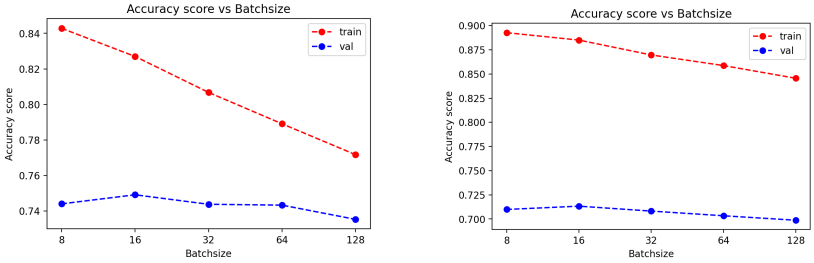
In Case 1, we pick the top 3 performing algorithms (with respect to highest accuracy scores on test dataset for pretrained version) and show how their training and validation metrics are changing during the training phase. The top 3 algorithms are: LSTM, CNN and Bi-LSTM. The comparison of Epochs vs accuracy score and Batch size vs accuracy score is made for all the top 3 selected algorithms. The comparison is done by training the algorithm for [5,10,15,20] epochs and [8,16,32,64,128] batch sizes. The graphs show how the training score and validation scores change with change in epoch and batch size values. This comparison is made after finding the other optimal hyperparameters (activation functions and optimizers).

5.1 LSTM



(a) Pretrained embs. (b) Embedding layer

Figure 16: LSTM: Epochs vs Accuracy score

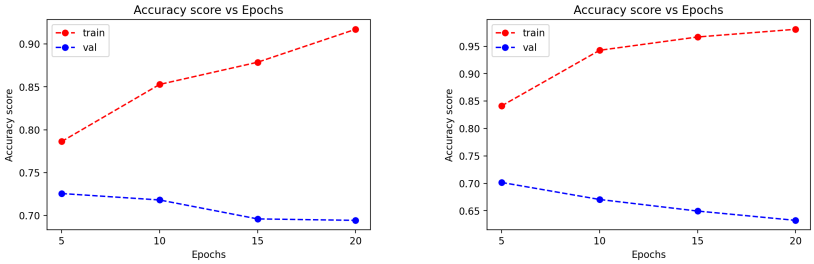


(a) Pretrained embs. (b) Embedding layer

Figure 17: LSTM: Batch size vs Accuracy scores

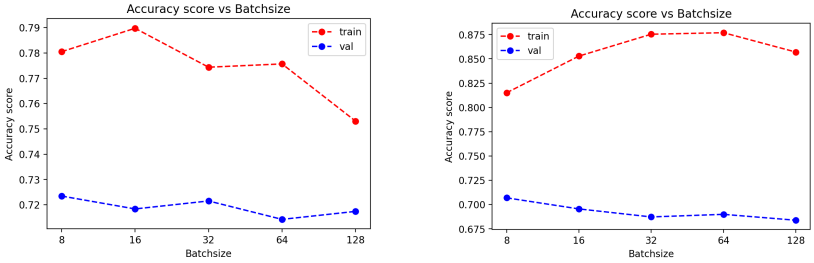
From the figures 16 and 17, the train accuracy increases with the increase in the epochs (for both pretrained version and the embedding layer version) and there is a decrease in the validation accuracy with increase in epochs. But with the increase in batch size, we see a sharp drop in training accuracy for the pretrained version. Finally, we can say that the models perform better with low batch size and less epochs.

5.2 CNN



(a) Pretrained embs. (b) Embedding layer

Figure 18: CNN: Epochs vs Accuracy score

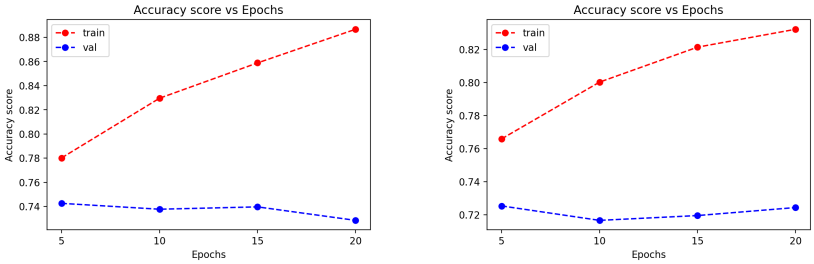


(a) Pretrained embs. (b) Embedding layer

Figure 19: CNN: Batch size vs Accuracy scores

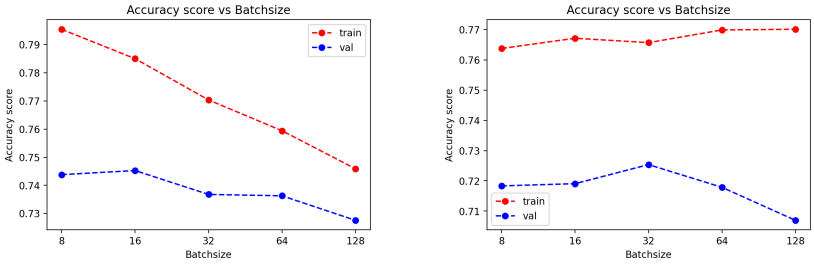
From the figure 18 and figure 19, the embedding layer version has the better performance in the training score but the pretrained version performs better with validation scores. This means the pretrained version better fits the model and has better accuracy at the test time also (from table 1). Also increasing the number of epochs is increasing the training accuracy but the validation accuracy decreases with increases in epochs. This pattern is similar in the case of change in batch size also.

5.3 Bi-LSTM



(a) Pretrained embs. (b) Embedding layer

Figure 20: Bi-LSTM: Epochs vs Accuracy score



(a) Pretrained embs. (b) Embedding layer

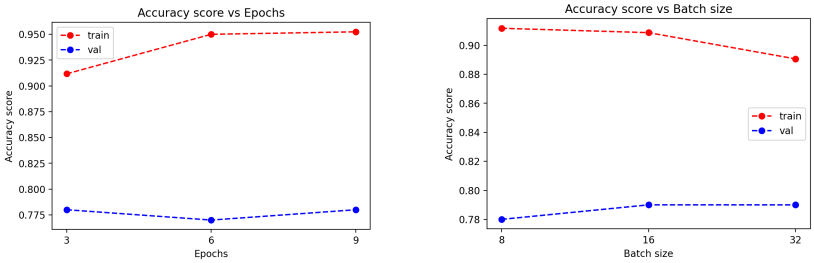
Figure 21: Bi-LSTM: Batch size vs Accuracy scores

Unlike the CNN and LSTM models, the validation accuracy doesn't decrease much with the increase in epochs but the best accuracy at test time is obtained with the model using 5 epochs and 16 batch size.

5.4 Transformer models

The analysis of transformer models is done in this subsection.

5.4.1 BERT

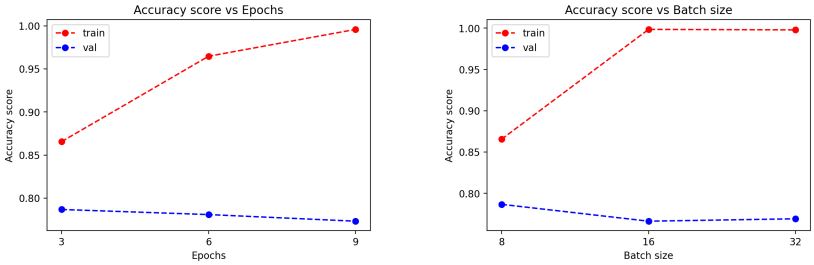


(a) Change in Epochs (b) Change in Batchsize

Figure 22: BERT: Analysis

From the tables 2 and 4, BERT is the best model with the highest accuracy scores obtained on the test set. From the graphs 22, the training accuracy increases (but not by a huge margin) as we increase the number of epochs. But it is the reverse for the increase in batch size where the training accuracy decreases and the validation accuracy increases. For BERT model, we found the best batch size is 32 and for other models it is either 8 or 16.

5.4.2 XLNet

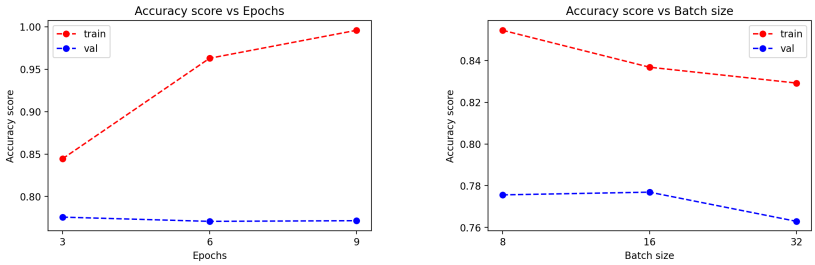


(a) Change in Epochs (b) Change in Batchsize

Figure 23: XLNet: Analysis

Similar to the other neural network models, XLNet also needs less epochs to make a generalisation of the data points. We can see that the training accuracy improves a lot with increase in epochs and batch size but there is not much change in the validation score.

5.4.3 RoBERTa



(a) Change in Epochs (b) Change in Batchsize

Figure 24: ROBERTa: Analysis

6 Optimal hyperparameters

The optimal hyperparameters are found for every neural network method which is shown in the following table. The hyperparameters used for pretrained version and the embedding layer version are almost same with 1 or 2 changes which are shown in the table. Most of the algorithms perform well with RMSprop as optimizer and with low batch sizes 16 and 8. (The accuracy scores shown here are for Case 1 pretrained version)

	Activation	Optimizer	Batch size	Epochs	Acc. Scores
MLP	relu	RMSprop	32	10 / 5	69.46
CNN	selu / elu	RMSprop	8	5	72.78
RNN	tanh	RMSprop	8	5	71.69
LSTM	tanh	adam	16	5	74.63
GRU	tanh	RMSprop	16	5	72.08
Bi-LSTM	tanh	adam	16	5	73.95
BERT	relu	adamW	32	3	78.13
XLNet	relu	adamW	8	3	77.98
ROBERTa	relu	adamW	8	3	77.03

Table 5: Optimal hyperparameters for Case 1

7 Discussions and Future work

The sentiment analysis gave us good results and the performance of the neural network models were as expected. Transformers gave the best results out of all the models. The neural networks were able to perform better on the selected text (which is a shorter form of the tweet containing the words that best describe the sentiment). By this, one idea would be to extract the important words from the tweets using tweet extraction methods and then use those words as input for sentiment analysis.

For the future work, we plan to:

1. Explore more models like RCNN, CNN-LSTM which are mix of convolution and recurrent networks

2. Experiment with transformer models and also other state of the art text classification models.

3. Study more on pretrained word embeddings other than GloVe embeddings.

8 Conclusion

From our experiments, we conclude that:

1. Transformers give the best performance of all the neural network models and BERT is the best model out of all transformer architectures we tested. Other than transformer models, LSTM is a good method which gives good performance at test time, thereby generalises the model well.

2. We observed that running the model of more number of epochs give good training scores and it keeps on increasing with increase in epochs but after a certain number of epochs (like 10 epochs) the validation scores are effected. This can mean that the model is starting to overfit by running it more number of times.

3. For the tweet dataset, 8 or 16 batch size is optimal for better scores at the test time. As we increase the batch size, the training accuracy drops, which can be a case of adding bias to the model or the model is underfitting the data points. Most of the neural network methods perform better on low batch sizes except BERT which provides good accuracy scores with relatively bigger batch size of 32.

4. More complex neural network architectures were able to perform better than the simple ones (like LSTM and transformers which give us better test scores). This means that the data points are fitted with a complex non-linear hyperplane which is possible to find using complex architectures.

5. RMSprop optimizer was the best among the optimizers we tested (relu, elu, selu, Adadelta and Adam) which worked well with the tweet dataset. And in overall, Pretrained version performed better than the embedding layer in most cases but the scores were very close to each other. The transfomer models use the AdamW optimizer which works better than adam optimizer.

6. Ranking of the neural network methods from best to worst is as follows (considering case 1 and the pretrained version):
$BERT > XLNet > RoBERTa > LSTM > Bi-LSTM > CNN > GRU > RNN > MLP$

7. In terms of running times, MLP has the least training times since it is a simple architecture and RNN takes the longest training times. (These time values can vary depending on the machine you are using for computation).

8. The accuracy and F1 scores calculated in Case 1 and Case 2 of our experiments are very close to eachother. This is because the slight imbalancy in the dataset is not affecting the overall performance of the model and hence we obtain good F1 scores.

A Appendix

A.1 Activation Functions

In this section, the activation functions used in the neural networks are explained in detail. The popular activation functions are discussed here:

1. Relu: Rectified Linear Unit is a function which outputs the same value if it is positive and outputs zero if it is negative.
   $f(x) = max(0, x)$
2. Sigmoid: It is an S-shaped curve and has a range between 0 and 1. Therefore, it is used to predict the probability of an output.
   $f(x) = \frac{1}{1+e^{-x}}$
3. Tanh: It is also an S-shaped curve but has a range of -1 and +1.
   $f(x) = tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
4. LeakyRelu: It increases the range of Relu function. The value of a is usually considered as 0.01.
   $f(x) = \begin{cases} ax, & \text{if } x < 1 \\ x, & \text{otherwise} \end{cases}$
5. Elu: Exponential linear unit, outputs negative values unlike relu. It has a constant value α.
   $f(x) = \begin{cases} \alpha(e^{x} - 1), & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases}$
6. Selu: Scaled Exponential Linear Unit, is a scaled version of elu with a scale factor of λ.
   $f(x) = \begin{cases} \lambda\alpha(e^{x} - 1), & \text{if } x < 0 \\ \lambda x, & \text{if } x \geq 0 \end{cases}$
7. Softmax: This activation is commonly used in the output later of a neural network. It outputs a probability for each class in a multi class setting, in which all the probabilities add to 1.
   $f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$ for i = 1,2,3,..K

A.2 Optimizers

Some of the optimizers used in our experiments are described here:

1. SGD: Stochastic gradient descent updates the parameter of loss function by using a step size. The equation is given as follows:
   $\theta = \theta - \eta\nabla_\theta J(\theta)$
   where $J(\theta)$ is the loss function of the example under consideration.
2. RMSprop: Root Mean Square Prop stores the values of past gradients in the form of exponentially weighted average.
   $s = \beta s + (1 - \beta)(\nabla_\theta J(\theta))^2$
   $\theta = \theta - \alpha\frac{\nabla_\theta J(\theta)}{\sqrt{s+\epsilon}}$where, s = exponentially weighted average of past squares of gradients

$\nabla_\theta J(\theta)$ = loss function gradient
β = hyperparameter which can be tuned
α = learning rate
ϵ = very small value to avoid dividing by zero

3. Adam: Adaptive Moment Estimation combines the idea of RMSprop and momentum. It computes exponential weighted average of past gradients and also the squares of past gradients. These values are corrected to remove the bias and then the weights are updated in the loss function.

$v = \beta_1 v + (1 - \beta_1)\nabla_\theta J(\theta)$
$s = \beta_2 v + (1 - \beta_2)(\nabla_\theta J(\theta))^2$
$v^{corrected} = \frac{v}{1-\beta_1}$
$s^{corrected} = \frac{v}{1-\beta_2}$
$\theta = \theta - \alpha \frac{v^{corrected}}{\sqrt{s^{corrected}}+\epsilon}$

where, v = exponentially weighted average of past gradients
s = exponentially weighted average of past squares of gradients
$\beta 1, \beta 2$= hyperparameters

A.3 Loss Function

The loss function relating to the multi class classification is discussed here:

Multi-Class Cross-Entropy Loss:
This loss function is used in the case of multi class classification problems and the softmax activation function is used with this.
$Loss = -\sum_{i=1}^{Outputsize} y_i \log \hat{y}_i$
where, $\hat{y}_i$ is the model prediction and the y_i is the actual label.

A.4 Additional graphs

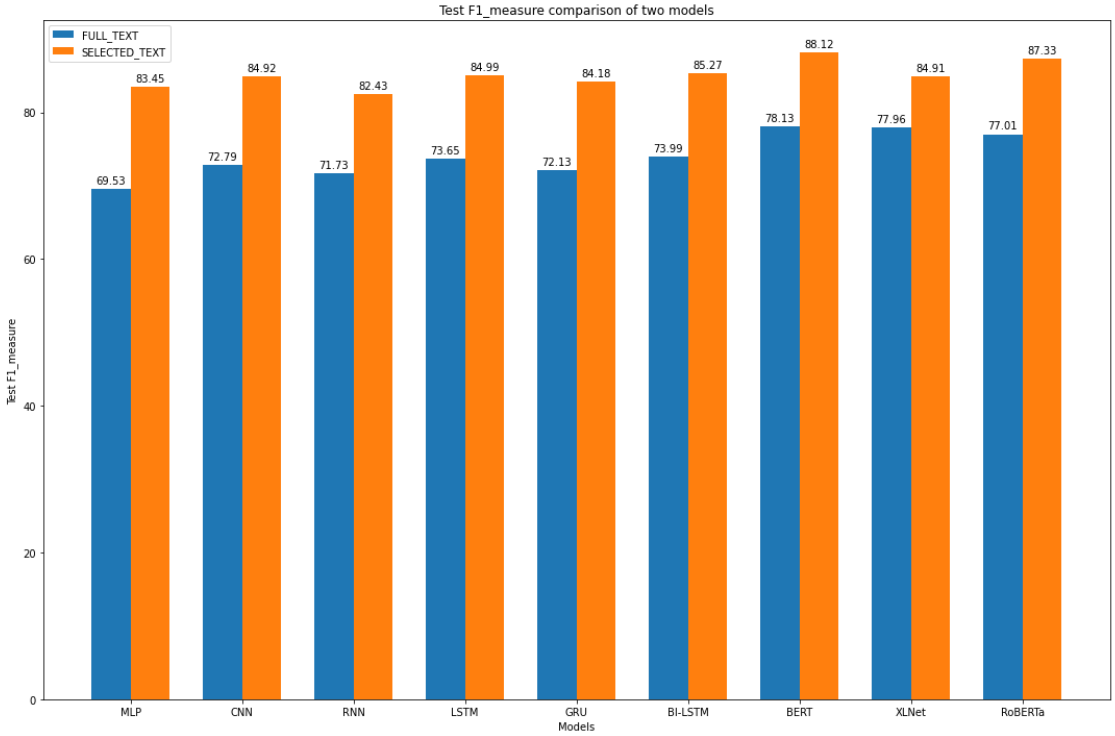


Figure 25: F1 measure comparison for different models

References

[1] Bert tweet extraction. https://www.kaggle.com/yutanakamura/dear-pytorch-lovers-bert-transformers-lightning.

[2] Bert, xlnet and roberta code. https://github.com/jinudaniel/amazon-fine-food-reviews.

[3] Convolutional neural networks for text classification. http://www.davidsbatista.net/blog/2018/03/31/SentenceClassificationConvNets/.

[4] Deep-dive into bidirectional lstm. https://www.i2tutorials.com/deep-dive-into-bidirectional-lstm/.

[5] Multilayer perceptron. http://naviglinlp.blogspot.com/2015/05/lecture-8-neural-network-word.html.

[6] Tweet sentiment extraction dataset. https://www.kaggle.com/c/tweet-sentiment-extraction/data.

[7] Understanding rnn. https://medium.com/x8-the-ai-community/understanding-recurrent-neural-networks-in-6-minutes-967ab51b94fe.

[8] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[9] A. K. I. S. Geoffrey E Hinton, Nitish Srivastava and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. 2012.

[10] S. Hochreiter and J. Schmidhuber. Long short-term memory. 143:1735–1780, 1997.

[11] A. Jacovi, O. S. Shalom, and Y. Goldberg. Understanding convolutional neural networks for text classification. *arXiv preprint arXiv:1809.08037*, 2018.

[12] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[13] V. Nguyen, T. Anh, and H.-J. Yang. Real-time event detection using recurrent neural network in social sensors. *International Journal of Distributed Sensor Networks*, 15:155014771985649, 06 2019.

[14] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[15] A. K. Uysal and S. Gunal. The impact of preprocessing on text classification. *Information Processing & Management*, 50(1):104–112, 2014.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[17] Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237, 2019.

[18] Y. Zhang, R. Jin, and Z.-H. Zhou. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, 1(1-4):43–52, 2010.