NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CZ4042 – Neural Networks & Deep Learning

Assignment 2
Text Emotion Recognition

Ahkshara Sankar	Bisakha Das	Soham Dandapath
U1823030A	U1823460E	U1822646A

1 Introduction

Text based emotion is a form of sentiment analysis that entails extracting and analyzing emotions from text. Text-based input is becoming a more frequent way for people to express their thoughts and feelings on a regular basis. With social media becoming increasingly widespread, text mining and analysis are at the forefronts of organizational success. Understanding the underlying emotions behind the concepts conveyed is critical for system improvement, and hence a burgeoning area of research in Natural Language Processing (NLP). These insights can be utilized in various domains such as Human-Computer Interaction (HCI), recommendation systems, online education, and psychology. However, accuracy is a prevailing challenge in the industry when it comes to text-based emotions.

Therefore, this project aims to investigate the performance of various approaches such as implementing the Bidirectional Encoder Representations from Transformers (BERT) Model, Global Vectors for Word Representation (GloVe) algorithm, and a combination of Convolutional Neural Networks (CNN) and Long-Short Term Memory (LSTM) in solving this conundrum.

In this research, an attempt has been made to capture the local information based on window-based approach using CNN while considering the global context using LSTM. To further improve the model pre-trained word embeddings are used that are specifically trained on tweets.

A BERT model based on transformers was also trained to compare the present state-of-the-art model with the models created from scratch. As evidenced by the outcomes of the studies, the transformer model is superior at collecting context at both the local and global levels. In addition, the research compared the conventional machine learning models such as Naïve Bayes Support Vector Machine (NBSVM), to neural network models, finding that neural network models outperform the conventional non-neural based models.

2 Background Research/Literature Review

2.1 Neural Network Embeddings

Neural network requires the categorical variable to be encoded in a numerical representation. Earlier methods involved the usage of one-hot representation of the categorical values. However, it produced sparse matrix which was inefficient both in terms of the size as well as the information it was capturing. All the values of the variable were assumed to be equidistant from each other. To resolve the above issue, word vectors [1] were introduced. Embeddings of the words that are learnt are continuous vector representations of words. The usage of embeddings reduces the dimensionality of categorical variables and enables them to be represented in a substantial manner in the transformed space.

2.1.1 Global Vectors for Word Representation (GloVe)

GloVe [2] is an unsupervised learning approach for obtaining word vector representations. It's a count-based model that uses a co-occurrence matrix to learn vectors. The concept is centered on the notion that ratios of word-to-word co-occurrence probabilities can store some type of meaning, which can subsequently be represented as vector differences. When the dot product of word vectors equals the logarithm of word likelihood of co-occurrence, the model learns them and performs well in word analogies and similarity tasks.

Below is the objective function GloVe uses to train word vectors from co-occurrence matrix:

$$J(\theta) = \frac{1}{2}\sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij})^2$$

GloVe embeddings takes into consideration the global level representation. The embeddings trained using GloVe algorithm on large scale dataset such as twitter and general corpus are openly available making it easier to be directly used in the model.

2.2 Models from Scratch

For the models that was created from scratch, the following types of neural architecture were employed.

2.2.1 Convolutional Neural Networks (CNN)

CNN has been extended to text-based categorization after it was first developed for image recognition. In general, feature extraction for CNN is done by sliding over input data to extract features and then applying. An activation function is applied to the output after numerous feature maps have been generated. Pooling layers are used to minimize dimensionality complexity while retaining considerable information from the convolutions, preventing overfitting. The usage of CNN in NLP tasks have been shown in [3].

The convolution layers in the NLP tasks are usually 1 dimension in nature. It takes into consideration the context of the word by using a sliding window across the sentence. It is faster than a recurrent neural network since it can be parallelized. The convolution can either be done at the character level or word level. The convolution layer extracts the meaningful local (n-gram) features which are useful for the sentence level prediction.

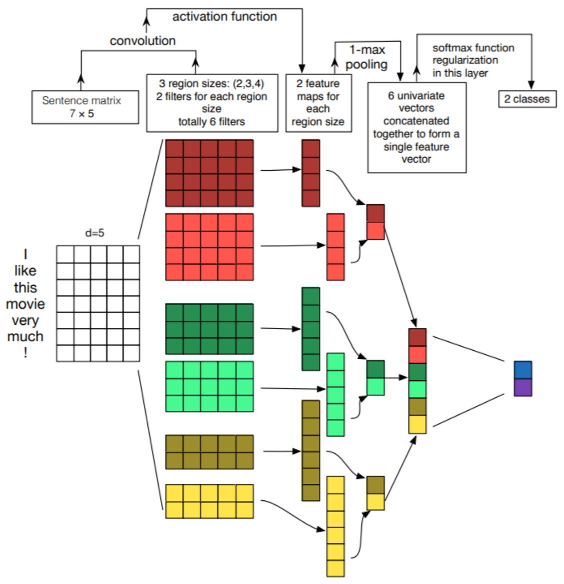

activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Figure 1 - CNN operation for NLP tasks - adapted from [5]

Figure 1 shows an example of a model that uses the CNN for the NLP task. The above is a word level encoder that extracts the context information using the window.

2.2.2 Long Short-Term Memory (LSTM)

LSTM networks are a type of recurrent neural network (RNN) that addresses the short-term memory problems that plague recurrent neural networks. It can capture temporal information such as the context of the word. The actions that occur within a cell to let LSTMs pass on/forget information are the key distinction with LSTMs. LSTMs are primarily concerned with the state of the cells and the different gates in the network. The LSTM network's gates are all based on sigmoid functions. Each cell in the LSTM network has a forget gate that selects the information that should be retained from the previous steps, an input gate that determines the information that should be added, and an output gate that determines the next hidden state.

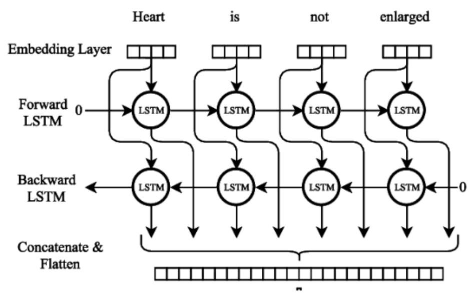


Figure 2 - Bidirectional LSTM in NLP – adapted from [6]

Bidirectional LSTMs as the name suggests, takes into consideration both the left and right context of a word as shown in Figure 2. This increases the model performance since the information from both the end are considered when making the sentence classification.

2.3 Pre-trained Models

Three pre-trained models are used for analysis. A brief introduction is provided below for each of the models.

2.3.1 Naïve Bayes – Support Vector Machine (NBSVM)

NBSVM [7] is a text-based classification strategy that combines Bayesian probability with a linear model like Support Vector Machine (SVM). This is commonly accomplished by substituting Nave Bayes log-count ratios for word count characteristics. NBSVM has been tested against a variety of strong text classification datasets and has shown to be a reliable approach as demonstrated in [7].

2.3.2 FastText

FastText [8,9] is a library developed by Facebook research to aid in the learning of word and text representations. It generates a sentence/document vector by averaging the word/n-gram embeddings. The sentence/document vector corresponding to the characteristics is then identified and classified using a multinomial logistic regression function. In addition, the probability distribution over pre-defined classes is obtained using a SoftMax layer. To implement efficient learning of word representations and sentence classifications, FastText was utilized.

2.3.3 Bidirectional Encoder Representations from Transformers (BERT)

BERT [10], a paper published by researchers at Google AI Language, focuses on applying a bidirectional training of Transformer, a popular attention model to language modelling. BERT pre-trains a deep bidirectional transformer from unlabeled text by fusing the left and right context using a Masked Language Model (MLM). MLM works by randomly masking a percentage of the tokens input and trying to predict the vocabulary ID of the masked word based on its context.

Implementing BERT can be broken down to 2 steps – pre-training and fine-tuning. During the pre-training stage, MLM kicks in and like a conventional language model, the hidden vectors corresponding to mask tokens are fed into an output SoftMax layer over the vocabulary. For the fine-tuning stage, the inputs and outputs from the previous stage are plugged in to tune all the parameters end-to-end.

BERT is a reliable model that has been pre-trained on a huge corpus of unlabeled text, such as the entire Wikipedia (2500 million words) and the Book Corpus (800 million words). Due to the ease of implementation and the robust model structure, it was chosen to be used in this research.

3 Dataset Description

This paper deals with 2 datasets – text_emotion.csv extracted from CROWDFLOWER and wassa data extracted from WASSA2017. To begin, some exploratory data analysis was performed in order to better comprehend the dataset.

3.1 WASSA

The WASSA dataset was categorized into 3 folders – train data, validation data and test data. In each folder, the data was categorized in separate text files based on its class. There is a total of 4 classes for this dataset – anger, joy, sadness, and fear. Each file was examined to determine the size of the various datasets (train, validation, and test). The statistics for the various datasets may be seen in the table below. To see the distribution of each class for each folder, a bar chart was plotted. Despite the fact that the sizes of each folder (train, validation, and test) differ, the distribution of classes is rather consistent, as seen in the figure below. The graphs also indicate that the data is evenly distributed between the classes.

Size of Train Dataset	3613
Size of Validation Dataset	347
Size of Test Dataset	3142
Total Number of Classes	4 in each dataset

Table 1 - Statistics of WASSA dataset

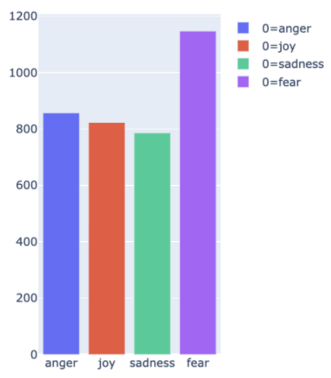


Figure 3 - Distribution of classes in Train Data

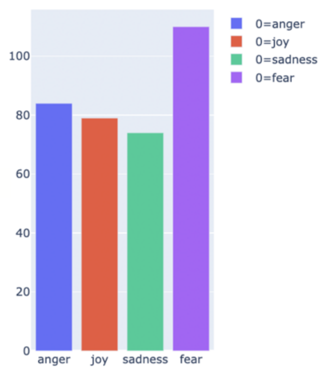


Figure 4 - Distribution of Data in Validation Data

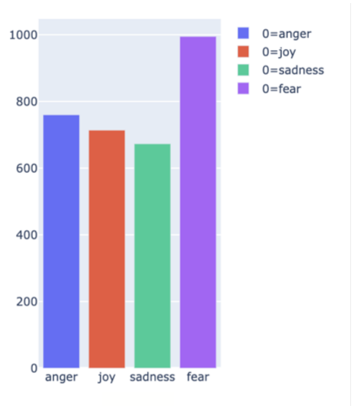


Figure 5 - Distribution of Data in Test Data

3.2 CROWDFLOWER

The size of the dataset was first determined by importing the *csv* file as a data frame and assessing the dataframe's size. The total number of classes in the dataset was then calculated using preliminary analysis. The dataset's identified statistics are shown in the table below.

Size of Dataset	40,000 tweets
Total Number of Classes	13

Table 2 - Statistics of text_emotion.csv dataset (CROWDFLOWER)

The distribution between the various classes was then shown using a bar chart, as seen below. The imbalance that exists between the different classes may be seen in the graph below. While the largest class ‘neutral’ contained 8638 tweets, the smallest class ‘anger’ contained 110 tweets. This is approximately a 98.62% difference in size.

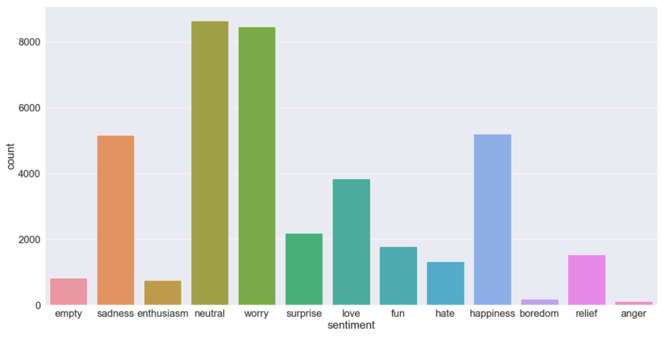


Figure 6 - Distribution of Data for text emotion

4 Methodology

Pre-processing

Despite the fact that the datasets were in different formats, the preprocessing was the same for each. The difference lied during the reading of dataset and converting it to a Pandas dataframe. The following is a summary of the significant actions that were performed to clean the data:

- Contraction Mapping: If a contraction is recognized, such as "ain't" or "aren't," it is substituted with its full/original form. For example, “ain’t” is replaced with “is not” and “aren’t” is replaced with “are not”.
- Parsing, a process of identifying tokens within a data instance and looking for recognizable patterns was achieved using the open-source software library named spaCy.
- The spaCy library was used to filter out punctuation, white space, numerals, and URLs. However, the content of the hashtags was retained.
- Unnecessary characters, such as single-syllable tokens and special characters, were eliminated.
- Spell Correction: A rudimentary spell correction was provided to deal with character repetition. Each character that appears more than twice in succession is condensed to two characters. For example, ‘Tooooooooooo Baddddddddddd’ will be shortened to ‘Too Badd’.
- Getting rid of the user who was specified. In most cases, tweets include other users' tags, such as "@das." However, because this is unrelated to determining the text's sentiment, it was eliminated.

The table below shows some examples exhibiting the pre-processing done.

Before preprocessing	**After preprocessing**
@tiffanylue i know i was listenin to bad habit earlier and i started freakin at his part =[	know be listenin to bad habit earlier and start freakin at his part
Layin n bed with a headache ughhhh...waitin on your call...	layin bed with headache ughh waitin on your call
Funeral ceremony...gloomy friday...	funeral ceremony gloomy friday
wants to hang out with friends SOON!	want to hang out with friend soo
mad the rain got me...now i cant go see jaiden *|)|/-\\\|\\\|/-\\*	mad the rain get me now can not go see jaiden
@niariley WASSUP BEAUTIFUL!!! FOLLOW ME!! PEEP OUT MY NEW HIT SINGLES WWW.MYSPACE.COM/IPSOHOT I DEF. WAT U IN THE VIDEO!!	WASSUP BEAUTIFUL follow me PEEP out my new hit single WWMYSPACECOM IPSOHOT I def WAT in the video

Table 3 - Examples of before and after Pre-processing

<u>Models from Scratch</u>

4 different model architectures were used for the analysis. The training was done from scratch using the training data. A comparison was also made between the usage of pre-trained embeddings and without them in a LSTM architecture. For all the other models, a pre-trained embedding was employed.

TextVectorization, a preprocessing layer adapted from Tensorflow Layers was used to map the text features in the dataset into integers. For the embeddings, the pre-trained GloVe[1] was used which was trained on 27 billion tweets with 100 dimensions as the output.

For the LSTM models, we use bidirectional. Depending on the model architecture, different layers were added. A summary of the model architecture is shown in the Appendix A.

<u>Pre-trained Models</u>

BERT, fastText and NBSVM were the three pretrained models implemented for text emotion recognition using Tensorflow Keras. The lightweight wrapper for Keras, ktrains, was used to build, train and deploy these three models. Ktrain's Bert specific preprocessing has been conducted for the Bert model, whereas standard preprocessing has been conducted for fastText and NBSVM.

<u>Tuning Learning Rates</u>

The learning rate governs the degree to which weights of the model are adjusted during training. Ktrain's lr_find() method has been used to first find a good initial learning rate. The plots below show the plots for finding a good learning rate for the WASSA dataset over BERT, fastText and NBSVM.

BERT

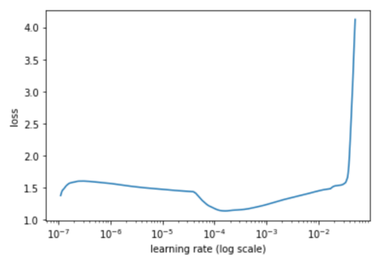


Figure 7 - Learning Rate vs Loss for BERT Model

fastText

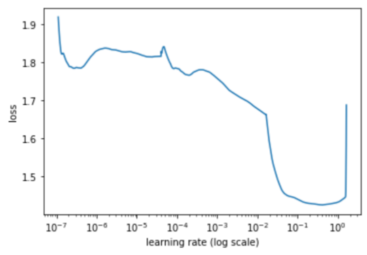


Figure 8 - Learning Rate vs Loss for fastText

NBSVM

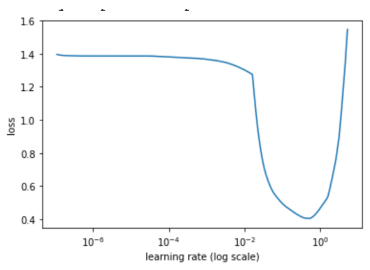


Figure 9 - Learning Rate vs Loss for NBSVM

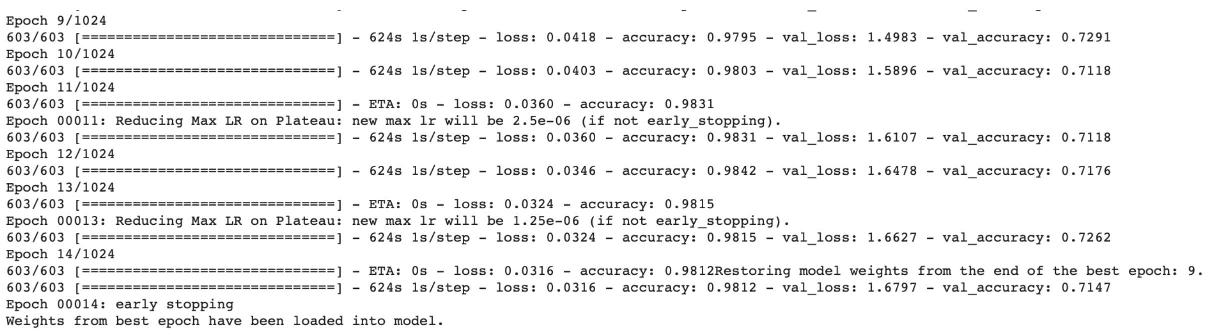
For the plots in the table above, the x-axis has learning rates in the log scale, and the y axis shows the loss encountered at that learning rate. For training the three models, appropriate learning rates have been selected depending on minimal loss, as observed in the respective loss-vs-learning rate plots.

<u>Training with Learning Rate Schedules</u>

After choosing a good initial learning rate, cyclic learning rates have been used, such that it is varied during training to help minimise loss and improve generalization. To allow for this, ktrain's autofit() method has been used with triangular learning rate policy from the paper titled "Cyclic Learning Rates for Training Neural Networks"[4].

[1] https://nlp.stanford.edu/projects/glove/

Early stopping with a patience level of 5 was used. Additionally, learning rate was reduced when the metric has stopped improving or the metric value has plateaued with a patience level of 2. The metric used was selected to be the accuracy on the validation data. An example training run with early stopping and reduce learning rate on plateau enabled, has been shown in the figure below.



```
Epoch 9/1024
603/603 [==============================] - 624s 1s/step - loss: 0.0418 - accuracy: 0.9795 - val_loss: 1.4983 - val_accuracy: 0.7291
Epoch 10/1024
603/603 [==============================] - 624s 1s/step - loss: 0.0403 - accuracy: 0.9803 - val_loss: 1.5896 - val_accuracy: 0.7118
Epoch 11/1024
603/603 [==============================] - ETA: 0s - loss: 0.0360 - accuracy: 0.9831
Epoch 00011: Reducing Max LR on Plateau: new max lr will be 2.5e-06 (if not early_stopping).
603/603 [==============================] - 624s 1s/step - loss: 0.0360 - accuracy: 0.9831 - val_loss: 1.6107 - val_accuracy: 0.7118
Epoch 12/1024
603/603 [==============================] - 624s 1s/step - loss: 0.0346 - accuracy: 0.9842 - val_loss: 1.6478 - val_accuracy: 0.7176
Epoch 13/1024
603/603 [==============================] - ETA: 0s - loss: 0.0324 - accuracy: 0.9815
Epoch 00013: Reducing Max LR on Plateau: new max lr will be 1.25e-06 (if not early_stopping).
603/603 [==============================] - 624s 1s/step - loss: 0.0324 - accuracy: 0.9815 - val_loss: 1.6627 - val_accuracy: 0.7262
Epoch 14/1024
603/603 [==============================] - ETA: 0s - loss: 0.0316 - accuracy: 0.9812Restoring model weights from the end of the best epoch: 9.
603/603 [==============================] - 624s 1s/step - loss: 0.0316 - accuracy: 0.9812 - val_loss: 1.6797 - val_accuracy: 0.7147
Epoch 00014: early stopping
Weights from best epoch have been loaded into model.
```

Figure 1110- Screenshot demonstrating Early Stopping Implemented

As seen in the case above, as soon as there is no improvement in the validation data accuracy, the maximum learning rate is reduced, and finally the weights of the epoch with the best val_accuracy is loaded into the model. The model architectures have been shown in Appendix A.

5 Results

The table below shows the performance of various models across the two datasets. The precision and recall are measured for the test dataset. Since the classes for CROWDFLOWER were highly imbalanced, it is important to use recall as a measure.

	Dataset 1 (WASSA)				**Dataset 2 (CROWDFLOWER)**			
Model	Train Accuracy	Test Accuracy	Precision	Recall	Train Accuracy	Test Accuracy	Precision	Recall
Embedding + CNN	0.49	0.59	0.59	0.59	0.34	0.33	0.28	0.34
LSTM	0.94	0.56	0.58	0.56	0.33	0.34	0.27	0.34
Embedding + LSTM	0.91	0.61	0.62	0.61	0.61	0.37	0.37	0.34
Embedding + LSTM+ CNN	0.94	0.61	0.62	0.61	0.37	0.36	0.30	0.36
BERT	0.98	0.73	0.74	0.73	0.51	0.40	0.38	0.40
FastText	0.82	0.68	0.69	0.68	0.34	0.35	0.32	0.35
NBSVM	0.98	0.59	0.61	0.59	0.56	0.20	0.20	0.20

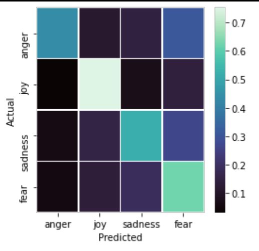
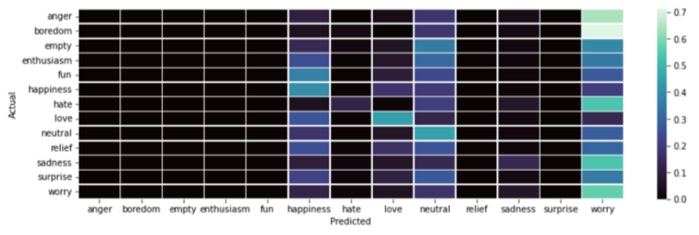
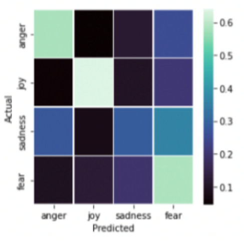
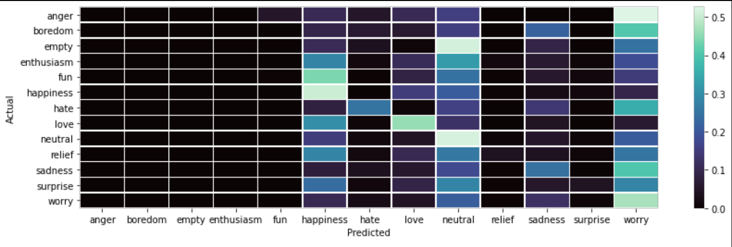
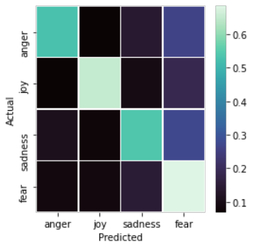
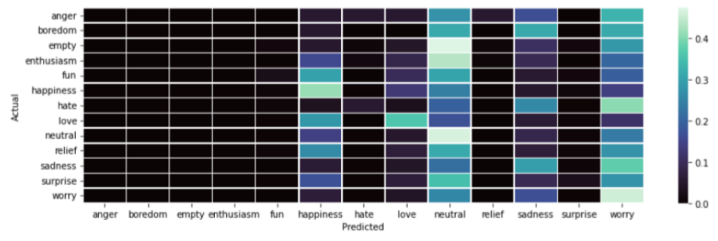
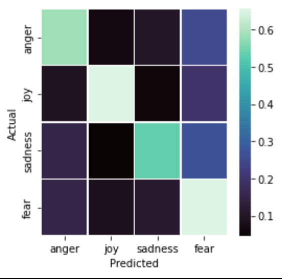
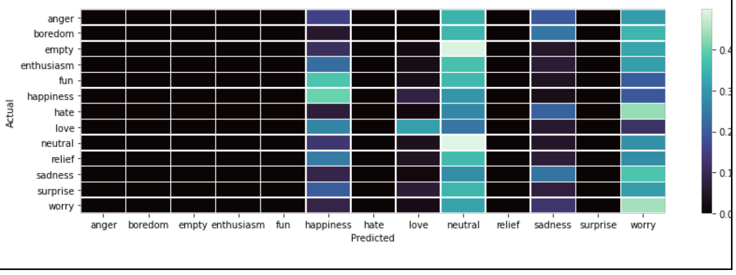
Table 4 - Summary of the Results Obtained

As seen in the table above, BERT outperformed all the other models. This can be explained by the robust nature and the state-of-the-art methodology BERT has. Models constructed from the scratch, such as Embedding + CNN, LSTM, Embedding+ LSTM, and Embedding + LSTM + CNN, fared much worse than the pre-trained models. While CNN and LSTM did not perform well on their own, combining the two yielded significantly better results. The combination of CNN and LSTM model was significantly stronger and performed lot better since CNN layers were utilized for feature extraction and LSTM was employed to enhance sequence prediction. The usage of LSTM particularly bidirectional layers allowed information based on context to be considered from previous sentence. In that respect, the CNN provided the regional level information while the LSTM took care of the global level context.

The pretrained models outperformed the models that were created from scratch, as they have already been trained on a large dataset and only required some fine-tuning for solving the problem at hand. These findings support the ease of use and remarkable accuracy of pre-trained models.

Another noteworthy finding is the difference in model performance between the two datasets. In comparison to the CrowdFlower dataset, all of the models performed substantially better with the WASSA dataset. This is attributable more to the dataset's nature than to the model itself. The CROWDFLOWER dataset with 13 classes was extremely imbalanced. The models were unable to attain high classification accuracy due to the imbalance in classes. Thus, only those with large training samples were being learnt while the others were being totally ignored as shown in the heatmaps displayed below.

The best and poorest classified classes were shown using heatmaps based on the confusion matrix derived for each model. The heatmap created for each model with each dataset is shown in the table below. In general, as shown in the table below, all the classes in the WASSA dataset were reliably categorized. However, majority of the classes in the CROWDFLOWER dataset, such as anger, boredom, and emptiness, were poorly categorized. This again reinforced the imbalance in dataset as the anger class contained only 110 tweets which is 98.62% lesser than the tweets present in the neutral class, further explaining the better results obtained for classes such as worry, happiness and neutral.

MODELS	WASSA	CROWDFLOWER
Embedding + CNN		
LSTM		
Embedding + LSTM		
Embedding + LSTM+ CNN		

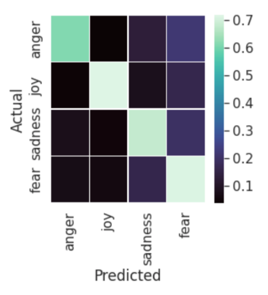
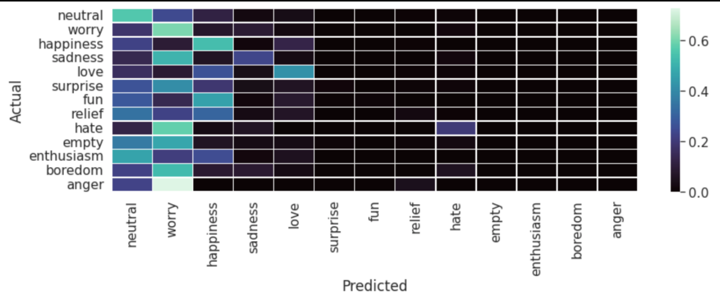
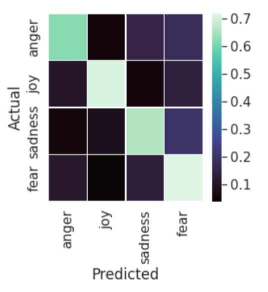
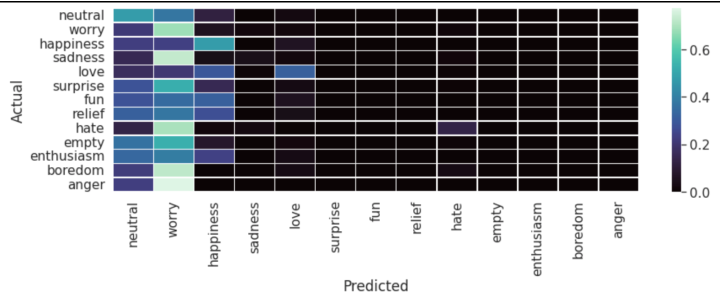
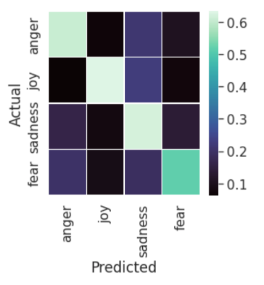
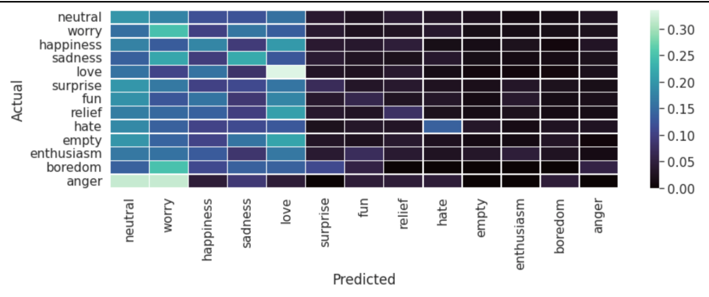
BERT		
FastText		
NBSVM		

Table 5 - Summary of Heatmaps created for each model

6 Conclusion

In this paper, a comparison was done among various neural network models that exists for text emotion recognition task. Primarily two sets of neural network models are used. One that are trained and built from scratch with the help of per-trained embeddings and the other is the pre-trained model architecture. Naturally, the pre-trained neural network architecture performed better in comparison to the models that have been trained from the scratch. However, NBSVM, a non-neural network approach performs worse than any of the neural network model demonstrating the suitability of the neural networks over traditional machine learning models in an NLP application. For the models trained from scratch, CNN and LSTM components were used to build the model and a synergy of both proved better than the individual component. The state-of-the-art models such as BERT uses attention model that are more recent and faster than the standard LSTMs while doing feature extraction like the CNNs.

References

[1] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In Proceedings of ICLR Workshops Track, 2013. arxiv.org/abs/1301.3781.

[2] J. Pennington, R. Socher, and C. D. Manning. GloVe: Global vectors for word representation. In EMNLP, 2014.

[3] Kim, Y. Convolutional neural networks for sentence classification, Conference on Empirical Methods in Natural Language Processing, pp. 1746–1751, 2014.

[4] Smith, L. N. (2017, April 4). *Cyclical learning rates for training neural networks*. arXiv.org. Retrieved November 14, 2021, arxiv.org/abs/1506.01186.

[5] Y. Zhang, B. C. Wallace, A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification, 13 Oct. 2015.

[6] S. Cornegruta et al, Modelling Radiological Language with Bidirectional Long Short-Term Memory Networks, 27 Sep. 2016. arxiv.org/abs/1609.08409

[7] S. Wang and C. D. Manning, "Baselines and bigrams: simple, good sentiment and topic classification," In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2 (ACL '12). Association for Computational Linguistics, USA, 90–94, 2012.

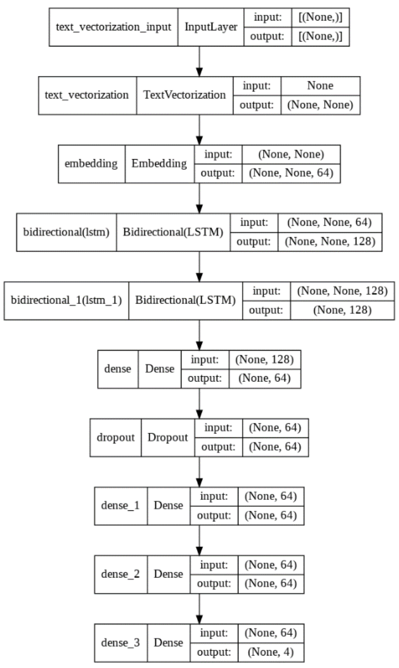
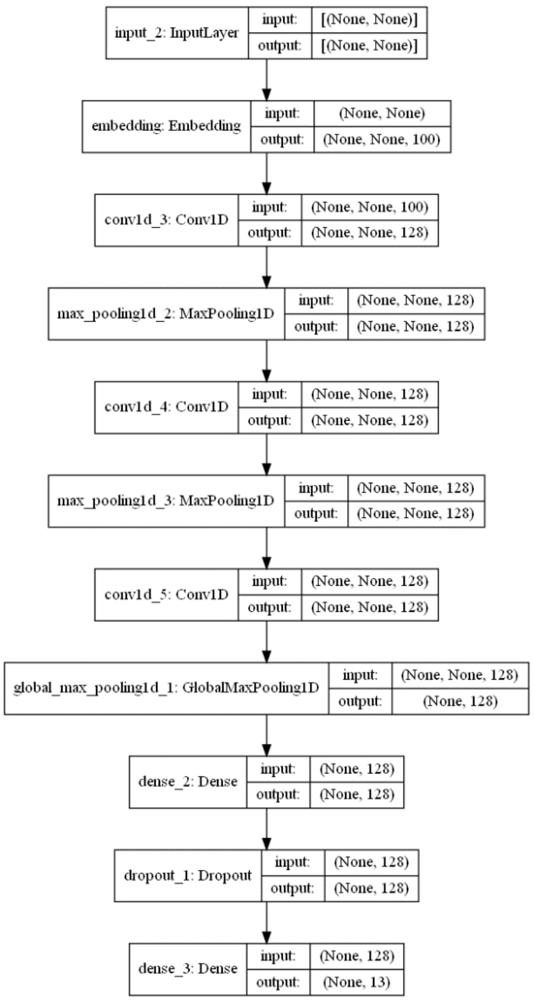
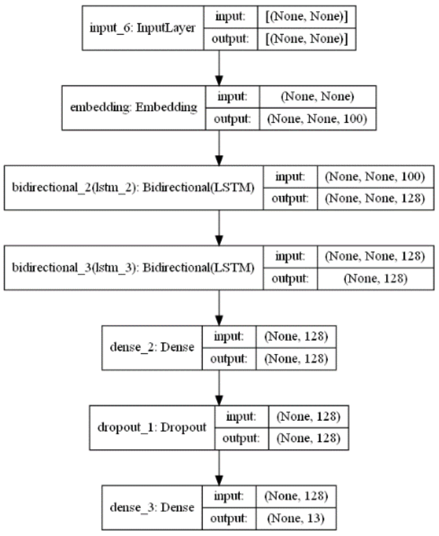
[8] Bojanowski, Piotr, et al. "Enriching word vectors with subword information." *Transactions of the Association for Computational Linguistics* 5 (2017): 135-146.

[9] Joulin, Armand, et al. "Bag of tricks for efficient text classification." *arXiv preprint arXiv:1607.01759* (2016).

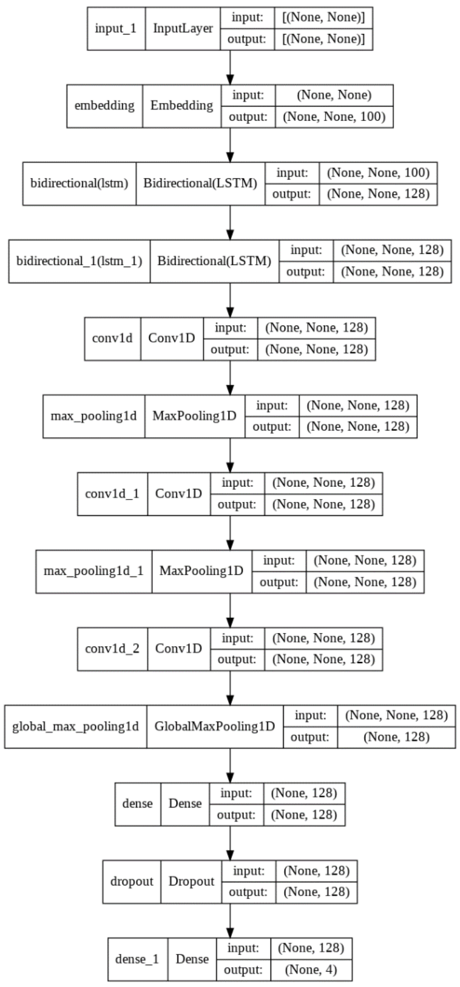
[10] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).

Appendix A – Model Architectures

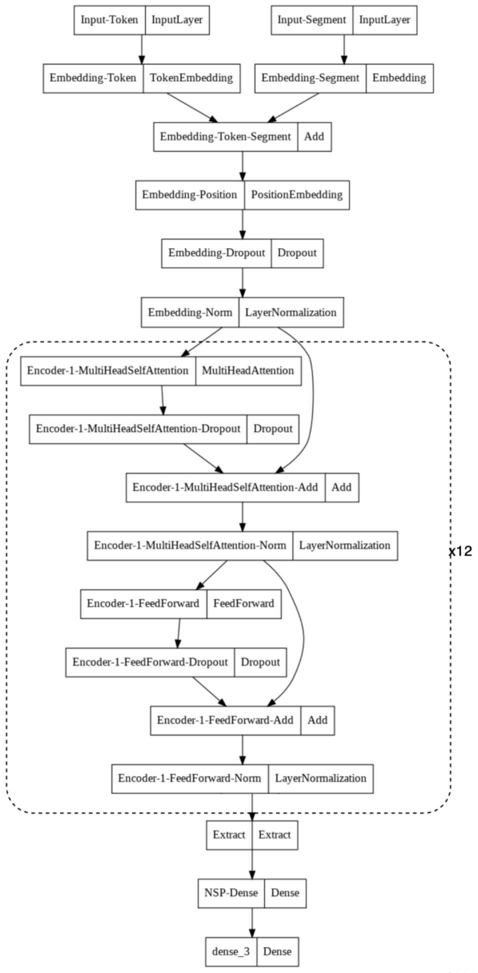
The table below summarises the model architectures. BERT uses twelve Multihead Attention Layers followed by dropout, layer normalization, feed forward and dropout. Instead of showing all the twelve encoder levels, only one has been shown in the diagram below, with a dotted box appearing around it to show that is the structure which gets repeated.

LSTM only	CNN + Embeddings
 text_vectorization_input | InputLayer | input: [(None,)] | output: [(None,)] text_vectorization | TextVectorization | input: None | output: (None, None) embedding | Embedding | input: (None, None) | output: (None, None, 64) bidirectional(lstm) | Bidirectional(LSTM) | input: (None, None, 64) | output: (None, None, 128) bidirectional_1(lstm_1) | Bidirectional(LSTM) | input: (None, None, 128) | output: (None, 128) dense | Dense | input: (None, 128) | output: (None, 64) dropout | Dropout | input: (None, 64) | output: (None, 64) dense_1 | Dense | input: (None, 64) | output: (None, 64) dense_2 | Dense | input: (None, 64) | output: (None, 64) dense_3 | Dense | input: (None, 64) | output: (None, 4)	 input_2: InputLayer | input: [(None, None)] | output: [(None, None)] embedding: Embedding | input: (None, None) | output: (None, None, 100) conv1d_3: Conv1D | input: (None, None, 100) | output: (None, None, 128) max_pooling1d_2: MaxPooling1D | input: (None, None, 128) | output: (None, None, 128) conv1d_4: Conv1D | input: (None, None, 128) | output: (None, None, 128) max_pooling1d_3: MaxPooling1D | input: (None, None, 128) | output: (None, None, 128) conv1d_5: Conv1D | input: (None, None, 128) | output: (None, None, 128) global_max_pooling1d_1: GlobalMaxPooling1D | input: (None, None, 128) | output: (None, 128) dense_2: Dense | input: (None, 128) | output: (None, 128) dropout_1: Dropout | input: (None, 128) | output: (None, 128) dense_3: Dense | input: (None, 128) | output: (None, 13) **Figure 12 12** - Model Summary for CNN + Embeddings
LSTM + Embedding	
 input_6: InputLayer | input: [(None, None)] | output: [(None, None)] embedding: Embedding | input: (None, None) | output: (None, None, 100) bidirectional_2(lstm_2): Bidirectional(LSTM) | input: (None, None, 100) | output: (None, None, 128) bidirectional_3(lstm_3): Bidirectional(LSTM) | input: (None, None, 128) | output: (None, 128) dense_2: Dense | input: (None, 128) | output: (None, 128) dropout_1: Dropout | input: (None, 128) | output: (None, 128) dense_3: Dense | input: (None, 128) | output: (None, 13)	

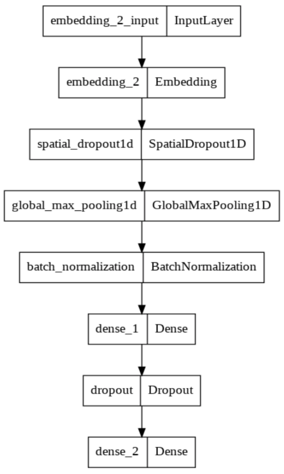
LSTM + CNN + Embedding


input_1 | InputLayer | input: [(None, None)] | output: [(None, None)]
embedding | Embedding | input: (None, None) | output: (None, None, 100)
bidirectional(lstm) | Bidirectional(LSTM) | input: (None, None, 100) | output: (None, None, 128)
bidirectional_1(lstm_1) | Bidirectional(LSTM) | input: (None, None, 128) | output: (None, None, 128)
conv1d | Conv1D | input: (None, None, 128) | output: (None, None, 128)
max_pooling1d | MaxPooling1D | input: (None, None, 128) | output: (None, None, 128)
conv1d_1 | Conv1D | input: (None, None, 128) | output: (None, None, 128)
max_pooling1d_1 | MaxPooling1D | input: (None, None, 128) | output: (None, None, 128)
conv1d_2 | Conv1D | input: (None, None, 128) | output: (None, None, 128)
global_max_pooling1d | GlobalMaxPooling1D | input: (None, None, 128) | output: (None, 128)
dense | Dense | input: (None, 128) | output: (None, 128)
dropout | Dropout | input: (None, 128) | output: (None, 128)
dense_1 | Dense | input: (None, 128) | output: (None, 4)

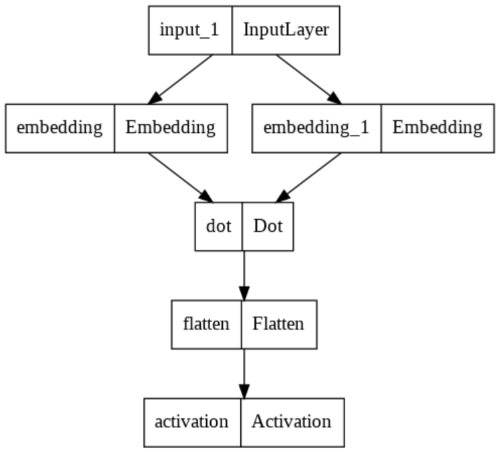
BERT


Input-Token | InputLayer
Input-Segment | InputLayer
Embedding-Token | TokenEmbedding
Embedding-Segment | Embedding
Embedding-Token-Segment | Add
Embedding-Position | PositionEmbedding
Embedding-Dropout | Dropout
Embedding-Norm | LayerNormalization
Encoder-1-MultiHeadSelfAttention | MultiHeadAttention
Encoder-1-MultiHeadSelfAttention-Dropout | Dropout
Encoder-1-MultiHeadSelfAttention-Add | Add
Encoder-1-MultiHeadSelfAttention-Norm | LayerNormalization
x12
Encoder-1-FeedForward | FeedForward
Encoder-1-FeedForward-Dropout | Dropout
Encoder-1-FeedForward-Add | Add
Encoder-1-FeedForward-Norm | LayerNormalization
Extract | Extract
NSP-Dense | Dense
dense_3 | Dense

fastText


embedding_2_input | InputLayer
embedding_2 | Embedding
spatial_dropout1d | SpatialDropout1D
global_max_pooling1d | GlobalMaxPooling1D
batch_normalization | BatchNormalization
dense_1 | Dense
dropout | Dropout
dense_2 | Dense

NBSVM


input_1 | InputLayer
embedding | Embedding
embedding_1 | Embedding
dot | Dot
flatten | Flatten
activation | Activation