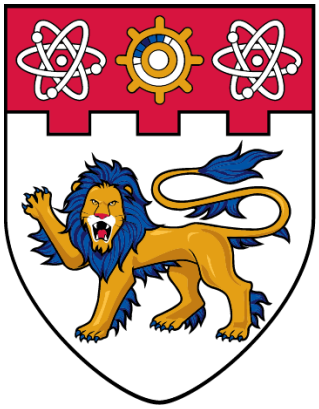


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Tweet Sentiment Analysis

CE4042: Neural Networks and Deep Learning

Couse Project

Kesarimangalam Srinivasan Abhinaya (U1923302J)

Singh Aishwarya (U1923952C)

Vedula Kartikeya (U1923891G)

School of Computer Science and Engineering

2021

Introduction

In recent times, social networking has become a huge part of our day-to-day activities. From WhatsApp to Instagram, we always stay connected and updated about the people whom we care about. One such social networking service is Twitter that has connected people from different parts of the world and supported numerous businesses. On this platform, users post and interact with messages known as “tweets”. Users are exposed to tweets that are of various categories such as excited, angry, sad, happy, etc. Thus, our team felt the need to identify the sentiment of a given tweet to help the user identify the type of content they are being exposed to on a daily basis. Hence, we decided to use the Long Short Term Memory networks to fulfil our mission.

Problem Statement

Text Emotion Recognition (TER): TER involves predicting emotions expressed in text and documents. Existing algorithms find emotion by learning the relationships of words using Recurrent Neural Networks (RNN) or Convolutional Neural Networks (CNN). RNN and CNN capture local information (i.e., emotion of words) and ignore the global information (i.e., emotion of sentence).

Our team decided to develop LSTM deep learning techniques to capture both local and global information. The local information refers to emotions expressed by words and global information refers to emotions expressed by the meanings of sentences.

Objective

The objective of this project, given the problem statement, is to seamlessly predict the sentiment of a given tweet/text input. The team aims to develop and implement a model that can help predict the emotion perceived from the text to an acceptable degree of accuracy, from the given dataset.

Evaluation Criteria

The team has decided to make use of loss, accuracy, and categorical_accuracy, given that the given problem statement is a classification problem (classify tweets into different sentiments), to evaluate the performance of the models designed, implemented, and tested.

Running Files

The code repository can be found via the following link - https://github.com/aish21/CE-CZ4042-Neural-Networks-and-Deep-Learning . The instructions for running the same are mentioned in the README file.

Data Preparation

To make the data useful for model architecture, implementation, and analysis, it needs to be prepared for preliminary exploration of the data. The given datasets have been used for this project –

1. **CROWDFLOWER**[15]: Dataset used for sentiment analysis to detect emotions in text. It contains 13 labels for the emotional content and each label corresponds to emotions such as happiness, anger, sadness, etc. There are hundreds to thousands of examples across these 13 labels
2. **WASSA2017**[16]: Dataset used in the research paper: deepCybErNet at EmoInt-2017: Deep Emotion Intensities in Tweet. It focuses on 4 emotions: anger, fear, joy, and sadness. It also has ratings for each of the aforementioned emotions (ranging from 0 to 1), given a sample tweet. The data is mainly split into 4 .txt files namely*: anger-ratings-0to1, fear-ratings-0to1, joy-ratings-0to1, and sadness-ratings-0to1.*

The following steps were implemented to prepare the given dataset:

- We decided to combine the two datasets so that we would have more data to train our model and make better predictions.
- As expected, there were a few different columns between the datasets which had to be taken care of before we passed it down to our model for training.
- Unnecessary columns in the CROWDFLOWER dataset such as: tweet_id and author, WASSA2017 dataset such as id and score, were removed as they can hamper model prediction.
- WASSA2017 had .txt files which had to be converted .csv format.
- Retained columns that were common for both datasets.
- Finally, after merging the 2 datasets, 15 categories were obtained namely: joy, happiness, enthusiasm, fun, sadness, worry, neutral, empty, hate, anger, fear, love boredom, relief, surprise.

The prepared dataset at the end of these steps can be found in the folder */data/Data-Cleaning-Ouput.csv.* With the data now prepared for use, the team decided to explore the data in depth. Understanding the data is an important step in the

construction of model architecture, as appropriate strategies can be employed based on the data given at hand to meet the needs of the problem statement and objective.

Exploratory Data Analysis

Exploratory Data Analysis is a detailed study designed to reveal the underlying structure of a data collection. It is significant because it reveals trends, patterns, and linkages that are not easily visible. One can't draw valid conclusions from a vast amount of data by just sifting through it; instead, one must examine it carefully and methodically via an analytical lens. Developing a sense for this vital information will assist in detecting errors, debunking misconceptions, and comprehending the links between many crucial factors. Such insights might finally lead to the selection of a suitable prediction model for sentiment analysis.

Statistical Analysis of Tweets

Tweet analytics visualizations are straightforward yet insightful tools. Some of these techniques have been implemented to gauge the scope of the problem statement, which are as follows - Character Frequency Distribution, Phrase Length Statistics, and Mean Word Size Distribution. These techniques are useful for exploring the core properties of tweet data.

First, character frequency distribution is implemented on the cleaned dataset. In this technique, the number of characters present in each phrase/sentence is obtained, which gives a rough estimation of tweet length, an important parameter in determining embedding dimensions for neural networks explained further in the report.

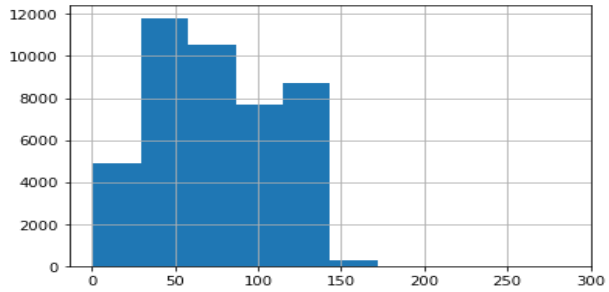


Figure 1: Character Frequency Plot

The histogram plot above indicates that tweets range from 0 to 200 characters, mostly between 30 and 70 characters. The next step is to analyse the number of words used in each tweet in the dataset.

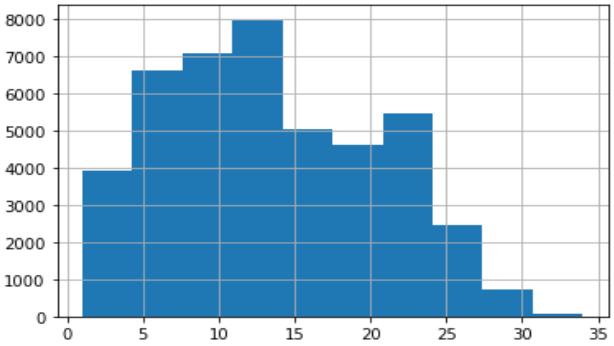


Figure 2: Word Frequency Plot

The plot above depicts that the number of words range between 2 to 35. This plot is consistent with the character frequency analysis as well, as both the plots put together also represent the character limit imposed by twitter for its users. This also implies that if the majority of the distribution lies within a small range of data, it may not be entirely

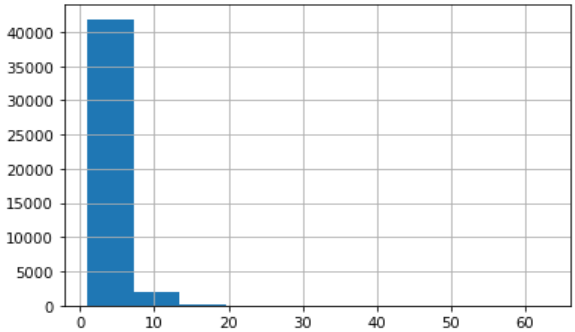


Figure 3: Average Word Length Plot

useful to use these statistical properties as predictor variables for tweet sentiment analysis. To further deep dive into statistical analysis, the next step is to obtain the average word length in each sentence –
The plot obtained for the dataset implies that although the range of average word length spans from 1 to 54, the distribution is extremely skewed to the left, which indicates the use of smaller words in tweets. This adds to the point of most words being similar in terms of length, which would therefore not be a good parameter for the prediction model.

Stopwords

The conclusion derived from the statistical analysis of tweets is not the best explanation for the skewed nature of the dataset. It cannot be assumed that all tweets are composed of smaller words. Therefore, another reason why *Figure 3,* here, is obtained is because of stopwords. Stop words are terms that are often filtered out before processing natural language. These are the most frequent words in any language (such as articles, prepositions, pronouns, conjunctions, and so on) that offer little information to the text. Analysing the number and kind of stopwords might provide useful insights into the data. The nltk library was used to obtain the corpus containing stopwords.

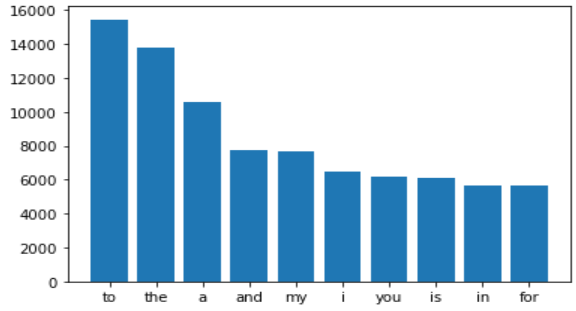


Figure 4: Stopwords Frequency Plot

The figure above shows the distribution of the occurrences of the most common stopwords found in the cleaned dataset, based on the NLTK library's list of stopwords in English. Therefore, the high occurrences of these stopwords provides a suitable explanation for the left-skewed nature of the average word length plot in *Figure 3*. To continue with the analysis, it is also important to understand which are the most frequently occurring words, apart from the stopwords, present in the dataset. To achieve this, the Counter function from the collections library is used to count and save the occurrences of each word in a list of tuples. When dealing with word-level analysis in natural language processing, this is a highly useful tool.

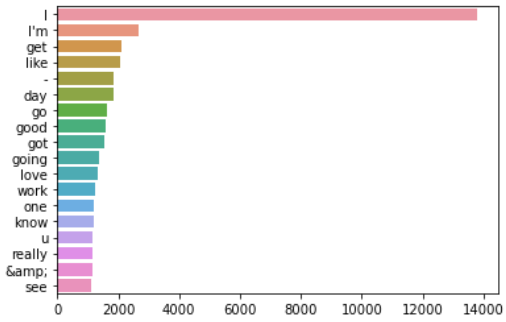


Figure 5: Non-Stopwords Frequency Plot

The plot above is the output of the aforementioned Counter function. It can be seen that along with some stopwords (keeping into account the capitalization and grammar of tweets) we have other words like 'love' and 'good' which could serve as differentiators for sentiment analysis of tweets.

N-gram Analysis

An n-gram is a continuous sequence of n elements from a particular text or audio sample. Depending on the process, the elements might be phonemes, syllables, letters, words, or base pairs. Often, n-grams are extracted from a text corpus. Looking at the most common n-grams will help to comprehend the context in which the term was employed. This can help with the selection of model architecture for sentiment prediction, as well as adjusting the model and its pertinent layers. The n-grams function from nltk.util has been used to construct n-grams.

Countvectorizer is used to create a representation of the language. It is used to convert a given text into a vector based on the frequency of each word in the full text. The value of each cell is just the number of words in that text sample. It is

a straightforward approach for tokenizing, vectorizing, and representing the corpus in an acceptable format. The following are the outputs of the top n-grams -

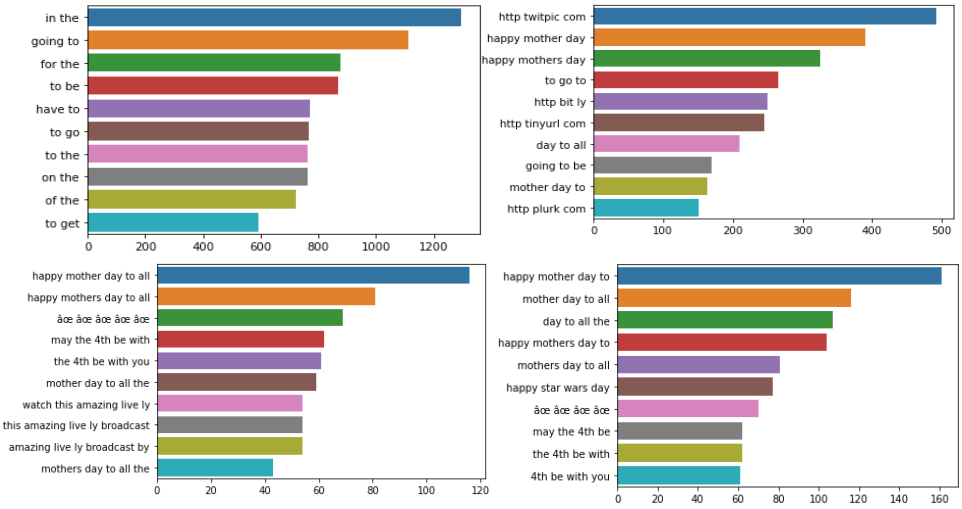


Figure 6: N-gram analysis (2,3,4,5 – gram)

The above plots are indicative of the fact that holidays and emojis are the most discussed topics in the dataset. To further analyse this Latent Dirichlet Allocation is used.

Latent Dirichlet Allocation (LDA)

Topic modelling is a technique for unsupervised categorization of texts that, like clustering on numeric data, discovers certain natural groups of things (topics) even when the user is unsure what to look for. Topic modelling enables huge datasets to be automatically organized, comprehended, searched, and summarized. It can aid in finding the dataset's hidden themes, categorizing the data according to the found topics. Latent Dirichlet Allocation (LDA) is a simple and effective topic modelling methodology. Each dataset is represented by a topic distribution, and each subject is represented by a word distribution. pyLDAvis is used to visualize the results of LDA interactively.

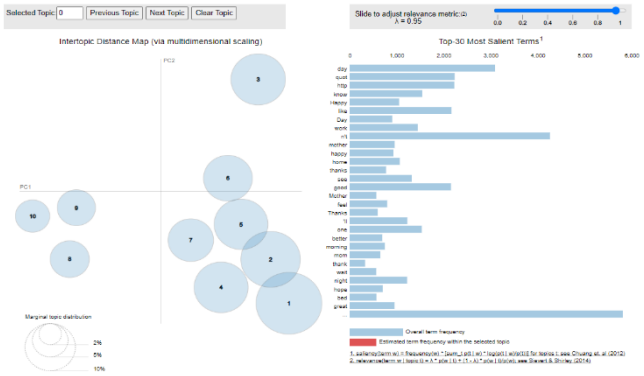


Figure 7: Interactive LDA exploration

The above figure is a snapshot of the interactive visualization of LDA. It can be tested out in 'Exploratory-Data-Analysis.ipynb', found in '\scripts'. The area of each circle on the left side shows the topic's prominence in relation to the corpus. The distance between the centres of the circles reflects how similar the themes are. The histogram of each topic on the right side displays the top 30 related terms.

Wordcloud

Wordcloud is an excellent technique to represent text data. The size and colour of each word in Wordcloud represent its frequency or significance. There are several settings that may be changed. Some of the most notable are stopwords, which denote the set of words that are not allowed to appear in the picture, and max words, which shows the maximum number of words to be displayed.

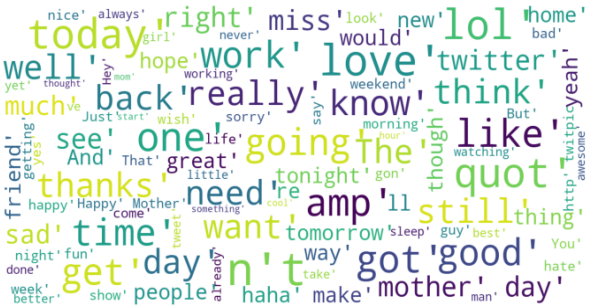


Figure 8: Wordcloud for dataset

Target Variable Analysis

It is important to know the distribution of the target variable so that appropriate feature engineering techniques can be employed to smoothen the distribution if it is skewed. If there is the frequency of some values are much higher than the others for the given dataset, there may not be enough training points for the remaining target variable values for the model to make accurate predictions.

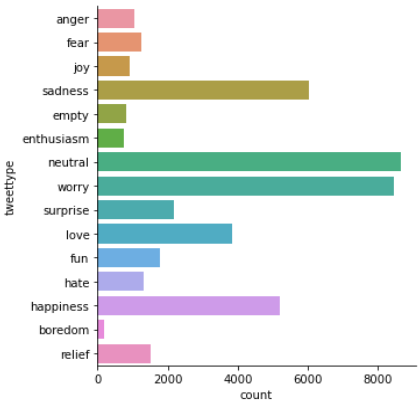


Figure 9: Target Variable Frequency Plot

The plot above shows the distribution of the target variables in the cleaned dataset. It can be seen, as described earlier, that the dataset appears to be quite skewed, and hence, appropriate feature engineering techniques will have to be deployed to strengthen the predictions of the model.

With the findings obtained from Exploratory Data Analysis, the next step is to tune the features to help aid the process of model testing and implementation.

Feature Engineering

As mentioned earlier, based on the information obtained from the previous section, it is imperative to engineer the features at hand before testing them in the prediction model, so that the skewed nature of the variables and other factors do not hamper its performance

Binning Categories

Continuing from the target variable analysis in Exploratory Data Analysis, it was concluded that there was a necessity to bin a few categories to improve the distribution of the variable values. This is achieved by binning all 'hate' categories as 'anger', 'happiness' as 'joy', 'fun' as 'enthusiasm', 'worry' as 'sadness' and 'empty' as 'neutral'. Therefore, the total number of categories and hence the outputs is brought down to 9 –

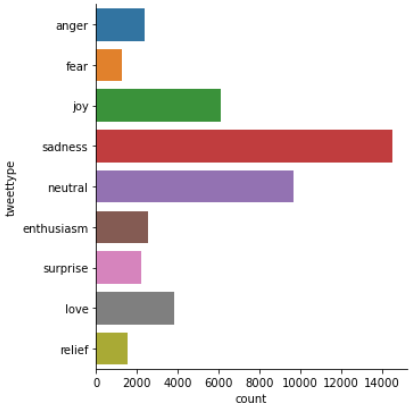


Figure 10: Target Variable Frequency Plot - Binned

Tweet Clean-up

Based on the findings of the Exploratory Data Analysis, this procedure is required. It was discovered that changes in capitalization, punctuation, and other grammatical differences might cause the model to interpret the same term differently. To address this, a number of procedures were taken to clean the tweets in the dataset. All tweets are lower cased, URLs, everything not a letter or apostrophe, usernames and ticker symbols are replaced with a space and single letter words are removed. For example, the tweet - '*Just joined #pottermore and was sorted into HUFFLEPUFF Ã°Å¸Ëœ ¡Ã°Å¸Ëœ ¡Ã°Å¸Ëœ ¡*' is cleaned to this - *'just joined pottermore and was sorted into hufflepuff*'.

Tokenization

Tokenization is the chore of splitting up a character sequence and a specific document unit into bits called tokens while potentially discarding some characters, such as punctuation. A token can be a word, a portion of a word, or just letters such as punctuation. The goal of tokenization is to provide a finite collection of symbols to the computer that it may combine to create the desired outcome.

Different methods of tokenization were tested on the cleaned-up tweets, which were - Use the python split() function, use regex to extract alphabets plus 's and 't, use NLTK word_tokenize() and use NLTK word_tokenize(), remove stop words, and apply lemmatization. After multiple experiments, the last method gave the best results in terms of evaluation metrics as well as model performance.

Lemmatization

Datasets will employ different versions of a term for grammatical reasons. There are also families of derivationally related words with comparable meanings. In many cases, it appears like a search for one of these terms will produce tweets that contain another word in the list. The purpose of lemmatization is to reduce a word's inflectional forms and, in certain cases, derivationally related forms to a single base form. Lemmatization is the process of doing things correctly by using a vocabulary and morphological analysis of words, with the goal of removing only inflectional ends and returning the base or dictionary form of a word.

Label Encoding

Label encoding is the process of translating labels into a numeric format so that they may be read by machines. Machine learning algorithms can then make better decisions about how those labels should be used. In supervised learning, it is a crucial pre-processing step for the structured dataset. In Python label encoding, the category value is replaced by a numeric value between 0 and the number of classes minus 1.

With the features now engineered, the team then decided to move on to model architecture, to construct and test different models for the given feature engineered dataset.

Model Architectures

In this section, the team experimented with different versions of LSTM and other models to choose the model that best suits the need of the problem statement. The plots for these architectures can be found in Appendix 1.

Why LSTM?

LSTMs have an edge over conventional feed-forward neural networks and RNN in many ways. This is because of the property of selectively remembering patterns for long durations of time. By moving from RNN to LSTM, we introduce more controlling parameters and gates, which control the flow and mixing of inputs as per trained weights. Thus, bringing more flexibility in controlling the outputs.

LSTM + Attention

The attention layer in Keras is based on the concept of cognitive attention. This effect enables the neural network to focus on important parts of input data and fade out the rest. The thought behind this mechanism is to direct the focus and great attention to a small but relevant part of the data. LSTMs enable RNN to remember inputs over a long time as it contains information in a memory. Due to this property, it is important that important parts of the sentence are highlighted when deciding so that we can pivot our attention to necessary details and efficiently use computing power over every sentence. The attention class takes in a layer and then performs the necessary functions such as initializing a keyworded variable length of arguments to the class, building the layer by adding the necessary weights and biases depending on the input shape and other functions to improve the layer, and then obtain the necessary outputs. The results obtained from this experiment were as follows:

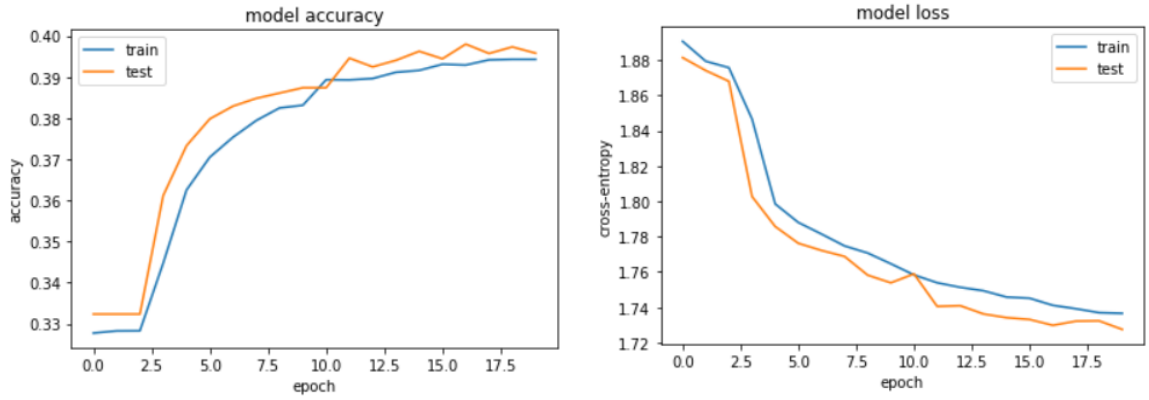


Figure 11: LSTM + Attention Accuracy and Loss Plots

LSTM + Bidirectional

Bidirectional LSTM or Bi-LSTM allows the neural network to have both backward and forward information about the sequence at every time step. Employing bidirectional will run inputs in two ways, one from past to future and one from future to past, thus preserving information from both past and future at any point in time. Using information from the future will make it easier for the network to understand what the next word is and thus predict the outputs better. We will need such a network because if we are able to predict the word in a sentence along with its context, then predicting its class would be much easier and we can improve accuracy.

1. Initial Bidirectional - We introduce one embedding layer and 2 bidirectional LSTM layers. The results obtained were as follows:

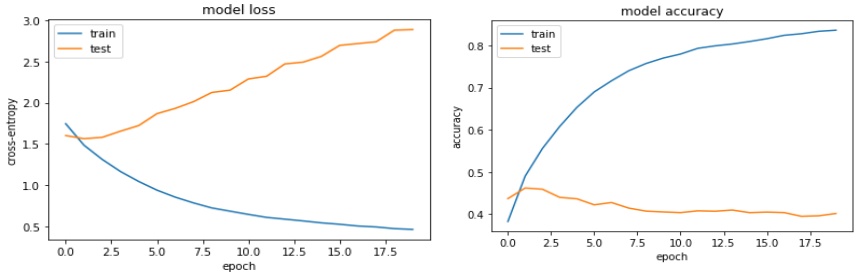


Figure 12: LSTM + Bidirectional (1)

2. Single Bidirectional - We introduce one embedding layer and one bidirectional LSTM layer. The results obtained were as follows:

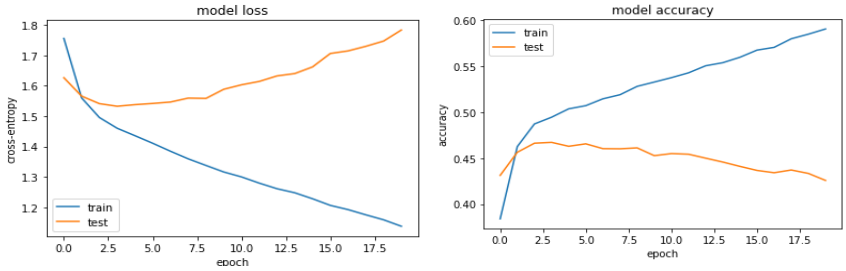


Figure 13: LSTM + Bidirectional (2)

3. Attention+Bidirectional+LSTM - In this experiment, we reuse the attention class we used in LSTM+Attention. We introduce an embedding layer and then pass the Bidirectional LSTM layer to the attention class to form the final attention layer. We then build our model based on this and the results thus obtained were as follows:

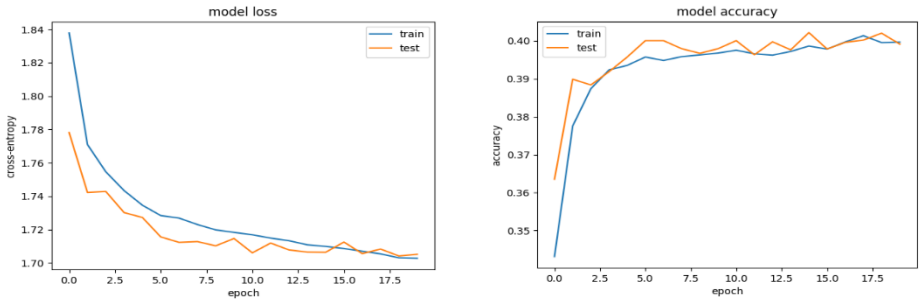


Figure 14: LSTM + Bidirectional (3)

Convolutional LSTM

The CNN LSTM architecture involves using Convolutional Neural Networks (CNN) layers for feature extraction on input data combined with LSTMs to support sequence prediction. CNN LSTMs class of models are both spatially and temporally deep and have the flexibility to be applied to a variety of vision tasks involving sequential inputs and outputs. The results of this experiment were as follows:

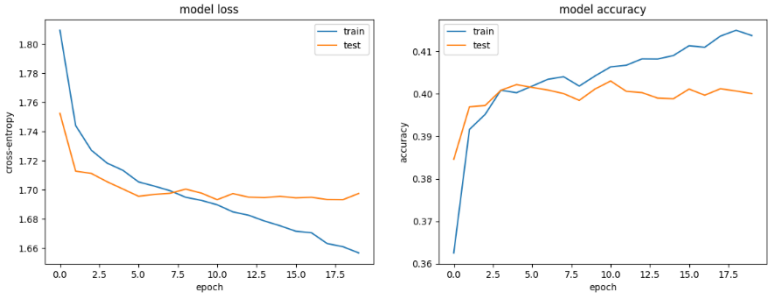


Figure 15: Convolutional LSTM

Vanilla LSTM

A Vanilla LSTM is an LSTM model that has a single hidden layer of LSTM units, and an output layer to make a prediction. We use a Basic LSTM with Spatial Dropout in this experiment. The results of these experiments were as follows:

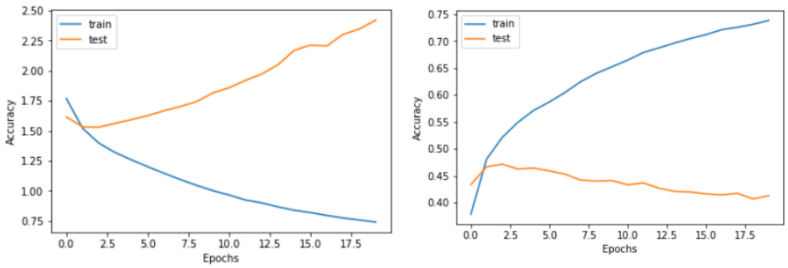


Figure 16: Vanilla LSTM

Hyperparameter Tuning

Throughout the course of this research, many models have been trained on the feature engineered dataset through various trials, and the one with the greatest performance will be selected. However, there is opportunity for improvement because no one can be certain that the model used is the best for emotion prediction. As a result, the goal is to enhance the model in every manner feasible. The hyperparameters of these models are an essential role in their performance; if adequate values for these hyperparameters are determined, the performance of a model can increase dramatically.

GridSearchCV

This method is useful for looping over specified hyperparameters and fitting the estimator (model) to training data. As a result, the best parameters from the list of hyperparameters may be chosen in the end. We pass predetermined hyperparameter values to the GridSearchCV method. We do this by constructing a dictionary in which we mention a certain hyperparameter as well as the values it can take. GridSearchCV examines the model for each combination of the values supplied in the dictionary using the Cross-Validation technique.

1. **Batch Size**: It is generally recognized that using a high batch size results in poor generalization. A smaller batch size has been proved to have faster convergence and allow the model to start learning before needing to view all of the data. The disadvantage of adopting a lower batch size is that the model will not always converge to the global optima. In this project, the values of batch size to choose from are 512, 256, 128 and 64. These values were selected based on the team's observation of data and the analysis of experiments performed with the different model architectures. The resulting optimum value for batch size came out to be 512.
2. **Number of Epochs**: Overfitting was one of the major issues encountered in the trials depicted and detailed above. When the number of epochs utilized to train a neural network model exceeds what is required, the training model learns patterns that are very unique to the sample data. As a result, the model is unable to perform effectively on a fresh dataset. In this project, the values of the number of epochs to choose from are 20, 50, 100, 200. These values were selected based on the team's observation of data and the analysis of experiments performed with the different model architectures. The resulting optimum value for the number of epochs came out to be 50.
3. **Optimization Algorithms:** Optimization is the process of iteratively training the model to provide a maximum and minimal function evaluation. Given the classification problem for sentiment prediction, it is critical that the correct optimization algorithm is chosen to suit the needs of the problem statement. During the conduction of experiments, the team believed that Adam optimizer would be the most suitable one for this problem, however, in the pursuit of performance improvement, the team then decided to ensure that the decision made was correct by putting this as a variable for GridSearchCV. The algorithms chosen were 'SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam'. The resulting optimization algorithm came out to be 'Adam'.
4. **Learning Rate:** The learning rate is a hyper-parameter that controls the pace at which an algorithm changes parameter estimations or learns parameter values. The learning rate determines how much weight is updated at the conclusion of each batch. In this project, the values of the learning rate to choose from are 0.001, 0.01, 0.03, 0.05, 0.07, 0.09, 0.1, 0.2. These values were selected based on the team's observation of data and the analysis of experiments performed with the different model architectures. The resulting optimum value for the learning rate came out to be 0.001.
5. **Network Weight:** Nodes in neural networks are made up of parameters known as weights, which are used to generate a weighted sum of the inputs. Weight initialization is the process of setting the weights of a neural network to tiny random values that establish the starting point for the neural network's optimization. In this project, the network weight initialization was done for the LSTM Bidirectional layers. The default value is 'glorot_uniform', however, the values that were fed into the GridSearchCV algorithm also included - 'uniform', 'lecun_uniform', 'normal', 'zero', 'glorot_normal', 'glorot_uniform', 'he_normal', 'he_uniform'. These values were selected based on the team's observation of data and the analysis of experiments performed with the different

model architectures. The resulting optimum value for the network weight initialization came out to be ‘normal’, thus different from the default hyperparameter value.

6. **Dropout Regularization:** Dropout is a regularization approach that approximates concurrent training of a large number of neural networks. During training, certain layer outputs are disregarded or "dropped out" at random. This has the effect of having each layer seem and be regarded as if it had a different number of nodes and connectedness to the preceding layer. In this project, the values of the dropout regularization to choose from are 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 and weight constraints are 1, 2, 3, 4, 5. These values were selected based on the team’s observation of data and the analysis of experiments performed with the different model architectures. The resulting optimum value for the dropout regularization came out to be 0.5 with a weight constraint of 4.
7. **Embedding Size and Input and Output Dimensions:** These hyperparameter values were significant in addressing the overfitting problem faced in the experiments that were conducted. They were tuned via a trial and test method, where the team began with some starting values, based on the data and the results that were obtained from the experiments conducted. These values were incremented and decremented, based on the requirements and the observed improvements in the model performance. In the end, the values that were chosen are (128,8) for the embedding layer, and 256 and 128 units for the 2 LSTM layers.

Model Testing

Based on the experiments conducted and the hyperparameter values tuned, the team’s final chosen model architecture was LSTM + Bidirectional layer. After the various experiments conducted by the team with the various custom architectures, the team decided to proceed with the bidirectional LSTM as the main model and performed hyperparameter tuning on it as explained earlier. This model helped us tackle the issue of overfitting to a great extent and helped us achieve an accuracy of approximately 0.43. Although this accuracy was better than all the previously constructed model architectures, the accuracy was unable to touch a higher number due to a highly unbalanced dataset and less data points.

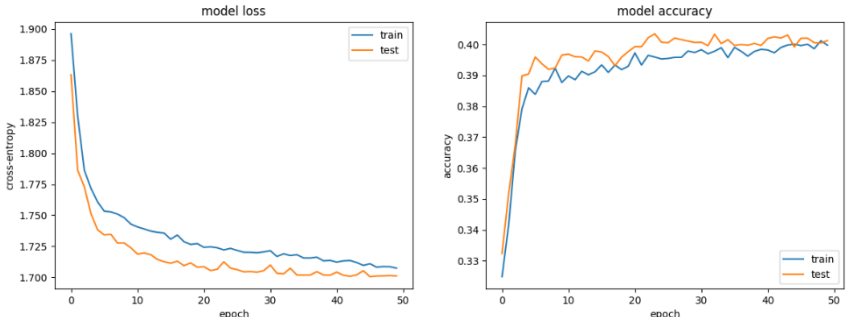


Figure 17: Final Model: 9 outputs

Postprocessing

Based on the results from Model Testing, it was evident to the team that despite the Herculean task to address and overcome the problem of overfitting in the dataset, it was becoming increasingly difficult to climb the accuracy score. One such reason the team felt was because of output dimensionality and the continued presence of too many categories and too few training data points for each category of the target variable in the dataset, as explained in Feature Engineering. To test this conjecture, as a part of postprocessing, the team decided to further bin the categories to positive, neutral, and negative, reducing the number of outputs to 3.

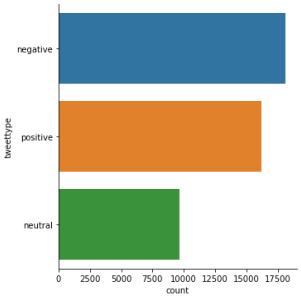


Figure 18: Postprocessing output

As it can be seen from the plot above, the graph is more evenly distributed, and hence the team was confident that the result would improve significantly with postprocessing

Result Discussion

Various experiments were conducted on models as explained in the model architecture. Due to uneven distribution of data points, it was really challenging to get a good prediction over all the classes. The main reason for this problem was the fact that most of the data points were under the category of sadness. Most of the times, the models were overfitting

due to this skewed data. With hyperparameter tuning, the value we got for the model testing turned out to be decent, but it was still based on accuracy. Still there was a slight overfitting with an accuracy of 0.4.

Based on the experiments conducted and the hyperparameter values tuned, the team's final chosen model architecture was LSTM + Bidirectional layer. After the various experiments conducted by the team with the various custom architectures, the team decided to proceed with the bidirectional LSTM as the main model and performed hyperparameter tuning on it as explained earlier. This model helped us tackle the issue of overfitting to a great extent and helped us achieve an accuracy of approximately 0.43. Although this accuracy was better than all the previously constructed model architectures, the accuracy was unable to touch a higher number due to a highly unbalanced dataset and less data points. After the postprocessing step, the postprocessed dataset was fed to the Bidirectional LSTM model. As expected, there was a significant improvement in the performance of the model. The model's categorical_accuracy improved by roughly 15%.

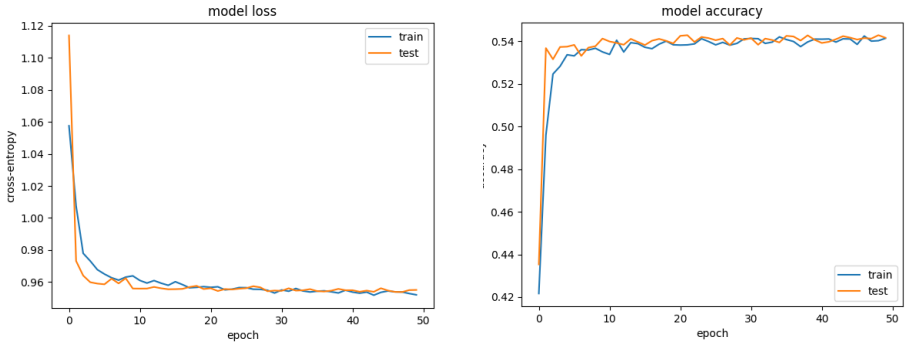


Figure 19: Postprocessed Model Results

Through the course of the different experiments conducted, involving tuning of hyperparameters, testing different model architectures, feature engineering the dataset variables, the team was able to illustrate a progression in model performance in each trial and experiment. By identifying the fallouts in each experiment and tweaking the issues to achieve a more reliable model, the team has displayed growth, progress, and significant improvements in model performance by deploying different deep learning techniques. The model progressed from having poor accuracy, not learning anything from data and overfitting in the 5th epoch to having a respectable accuracy, addressing the overfitting problem, and performing much better in terms of the evaluation metrics. The progression for the same can be seen in the outputs of all the experiments that were conducted through the course of this project in Appendix 2 and 3.

Scope for Improvement

To conclude the project, the team would like to elaborate on the areas where the project can be improved in order to achieve a more seamless end-to-end sentiment prediction model -

1. **Real time extension of the predictions** – A browser extension of the model, that can be used with Twitter, which gives real time prediction of the sentiment of the tweet typed out.
2. **Need for more data** – 40,000 data points, along with originally 15 target variables was not enough for the model to learn well, thus collection of more data can improve the performance of the model significantly, and give room for more sentiments as well, given it has enough training data points.
3. **Using a multi – input architecture to use more of the extracted features from the tweet** – In Feature Engineering, the team made use of the punctuation counts and other features extracted from the dataset. These features, too, can be used in the prediction model.

References

[1] "Understanding LSTM Networks -- colah's blog", Colah.github.io, 2021. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 1- Nov- 2021].

[2] P. Kumar and V. →, "An Introduction to N-grams: What Are They and Why Do We Need Them? - XRDS", XRDS, 2021. [Online]. Available: https://blog.xrds.acm.org/2017/10/introduction-n-grams-need/. [Accessed: 5- Nov- 2021].

[3] "NLP - Linguistic Resources", Tutorialspoint.com, 2021. [Online]. Available: https://www.tutorialspoint.com/natural_language_processing/natural_language_processing_linguistic_resources.htm. [Accessed: 13- Nov- 2021].

[4] "How To Remove Stopwords In Python | Stemming and Lemmatization", Analytics Vidhya, 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2019/08/how-to-remove-stopwords-text-normalization-nltk-spacy-gensim-python/. [Accessed: 12- Nov- 2021].

[5] "Using CountVectorizer to Extracting Features from Text - GeeksforGeeks", GeeksforGeeks, 2021. [Online]. Available: https://www.geeksforgeeks.org/using-countvectorizer-to-extracting-features-from-text/. [Accessed: 15- Nov- 2021].

[6] "Latent Dirichlet Allocation(LDA)", Medium, 2021. [Online]. Available: https://towardsdatascience.com/latent-dirichlet-allocation-lda-9d1cd064ffa2. [Accessed: 6- Nov- 2021].

[7] "Feature Extraction for Sentiment Classification on Twitter Data", International Journal of Science and Research (IJSR), vol. 5, no. 2, pp. 2183-2189, 2016. Available: 10.21275/v5i2.nov161677.

[8] "Tokenization", Nlp.stanford.edu, 2021. [Online]. Available: https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html. [Accessed: 2- Nov- 2021].

[9] "Stemming and lemmatization", Nlp.stanford.edu, 2021. [Online]. Available: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html. [Accessed: 25- Oct- 2021].

[10] "What is Label Encoding in Python | Great Learning", GreatLearning Blog: Free Resources what Matters to shape your Career!, 2021. [Online]. Available: https://www.mygreatlearning.com/blog/label-encoding-in-python/. [Accessed: 29- Oct- 2021].

[11]"Effect of batch size on training dynamics", Medium, 2021. [Online]. Available: https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e. [Accessed: 7- Nov- 2021].

[12] "Understanding Learning Rate in Machine Learning", GreatLearning Blog: Free Resources what Matters to shape your Career!, 2021. [Online]. Available: https://www.mygreatlearning.com/blog/understanding-learning-rate-in-machine-learning/. [Accessed: 8- Nov- 2021].

[13] J. Brownlee, "Weight Initialization for Deep Learning Neural Networks", Machine Learning Mastery, 2021. [Online]. Available: https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/. [Accessed: 9- Nov- 2021].

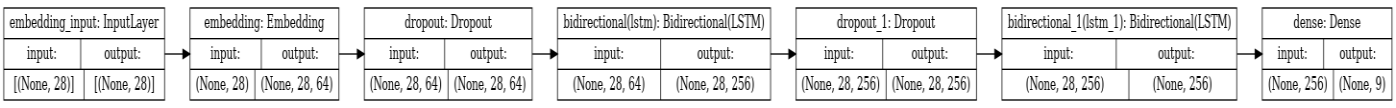
[14] J. Brownlee, "A Gentle Introduction to Dropout for Regularizing Deep Neural Networks", Machine Learning Mastery, 2021. [Online]. Available: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/. [Accessed: 10- Nov- 2021].

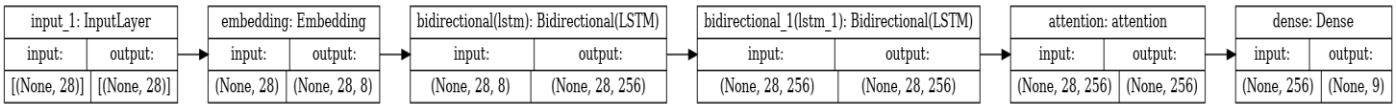
[15] “Sentiment analysis in text - dataset by Crowdflower,” data.world, 21-Nov-2016. [Online]. Available: https://data.world/crowdflower/sentiment-analysis-in-text. [Accessed: 15-Nov-2021].

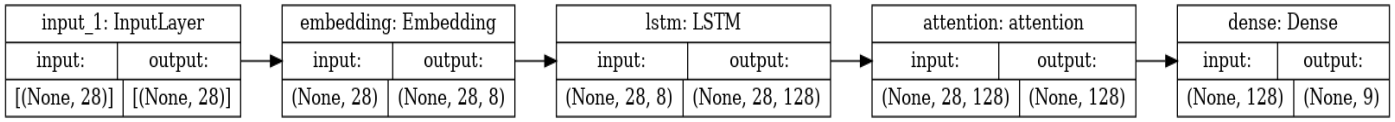
[16] Vinayakumarr, “Vinayakumarr/Wassa-2017: Determining Emotion Intensity,” GitHub. [Online]. Available: https://github.com/vinayakumarr/WASSA-2017. [Accessed: 15-Nov-2021].

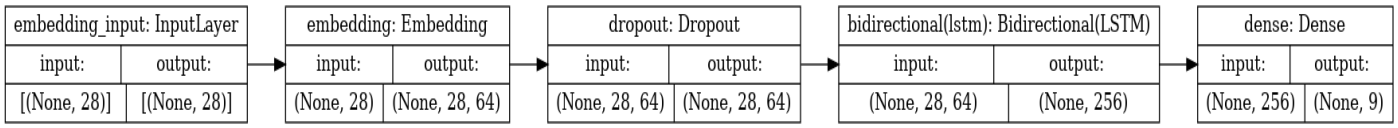
Appendix

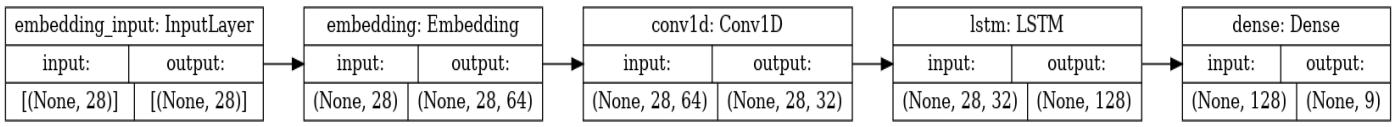
[1] Model Architectures – (Convolutional – 1D + Dropouts, Bidirectional, Bidirectional + LSTM + Attention, LSTM + Attention, Single Bidirectional)

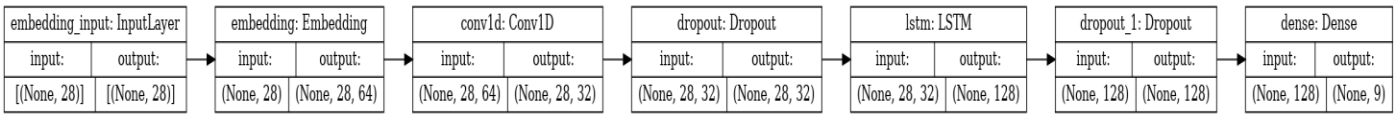




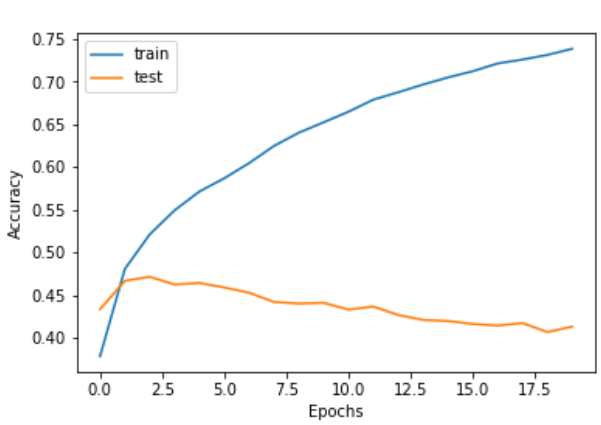


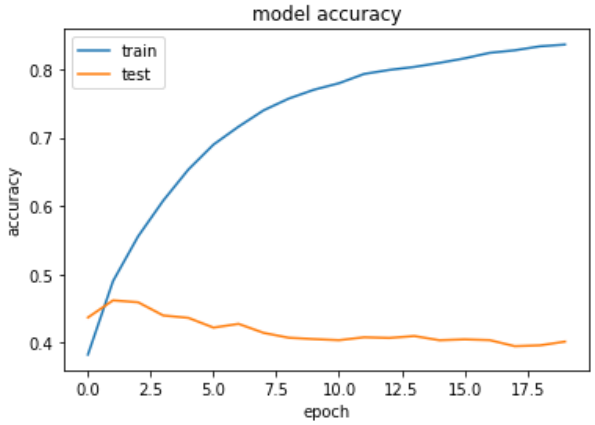


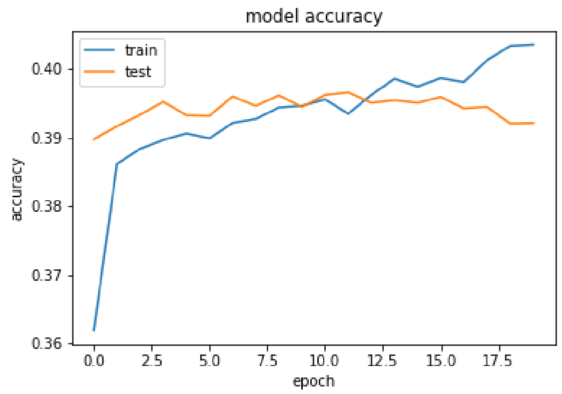


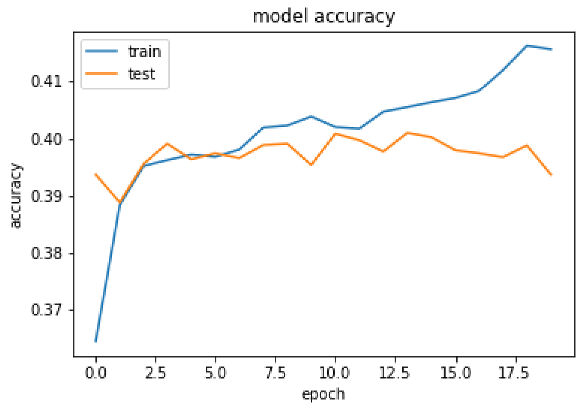


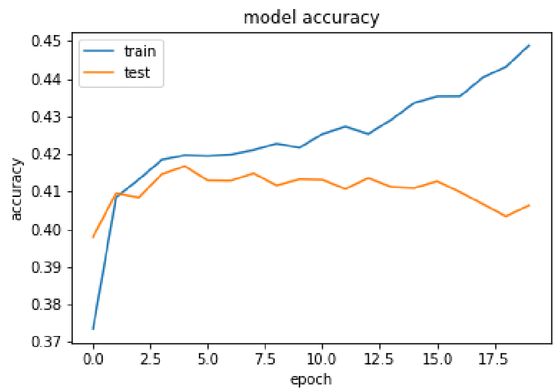
[2] Model Progression – All experiments conducted - Accuracy

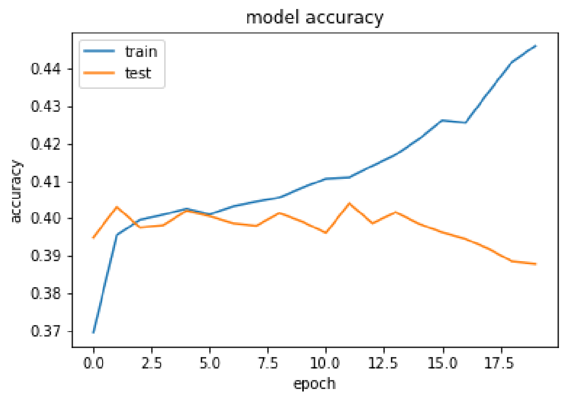


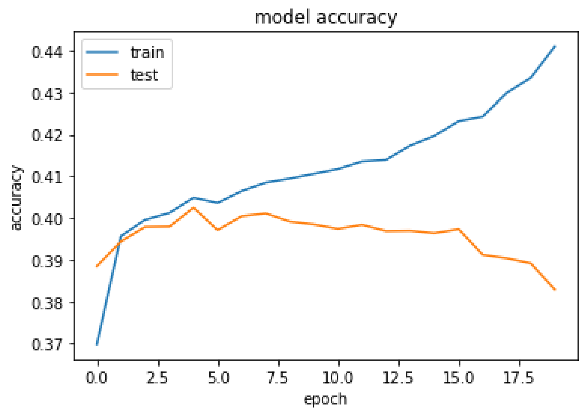


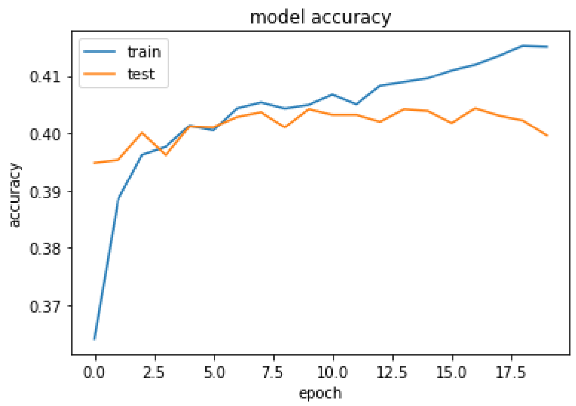

model accuracy
train
test
accuracy
epoch
0.36
0.37
0.38
0.39
0.40
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

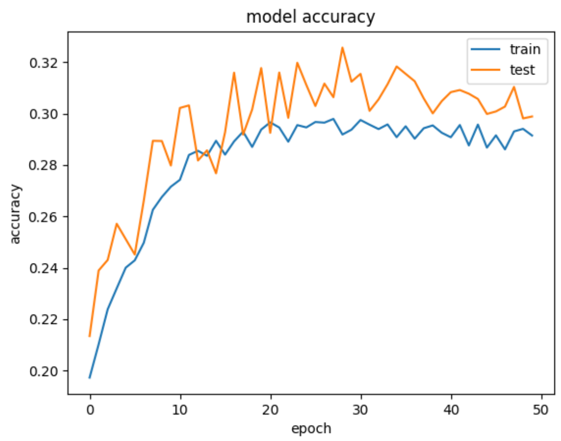

model accuracy
train
test
accuracy
epoch
0.37
0.38
0.39
0.40
0.41
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

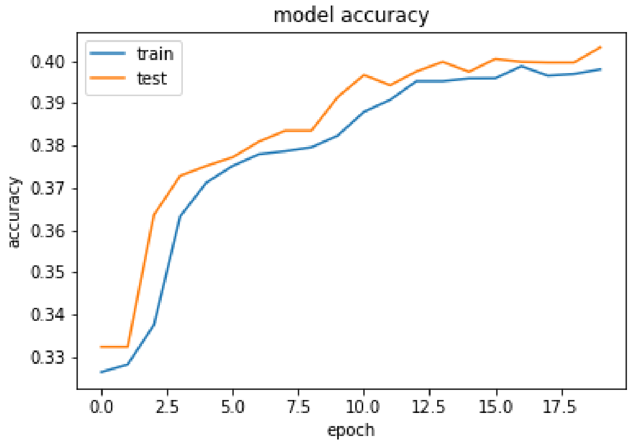

model accuracy
train
test
accuracy
epoch
0.37
0.38
0.39
0.40
0.41
0.42
0.43
0.44
0.45
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

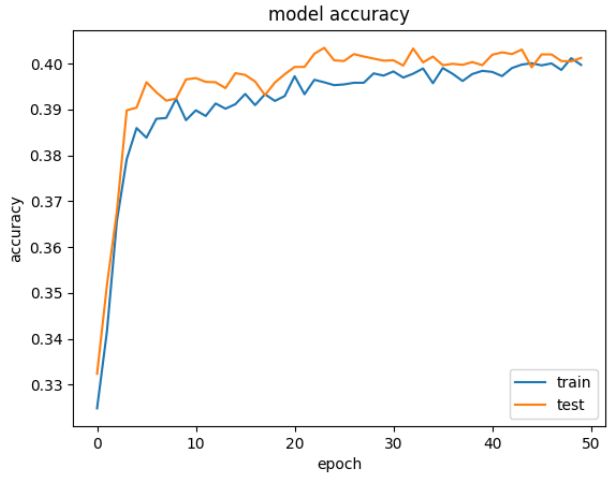

model accuracy
train
test
accuracy
epoch
0.37
0.38
0.39
0.40
0.41
0.42
0.43
0.44
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

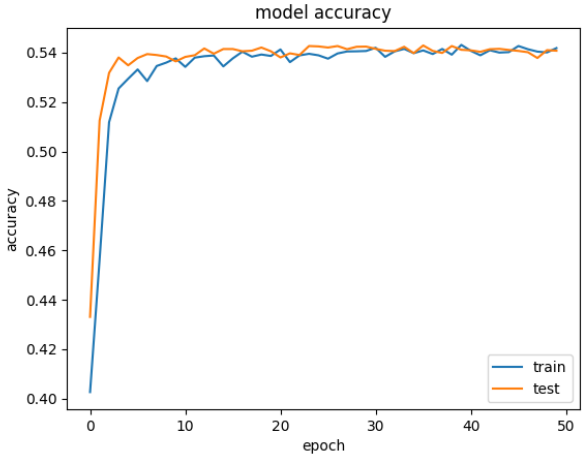

model accuracy
train
test
accuracy
epoch
0.37
0.38
0.39
0.40
0.41
0.42
0.43
0.44
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


model accuracy
train
test
accuracy
epoch
0.37
0.38
0.39
0.40
0.41
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

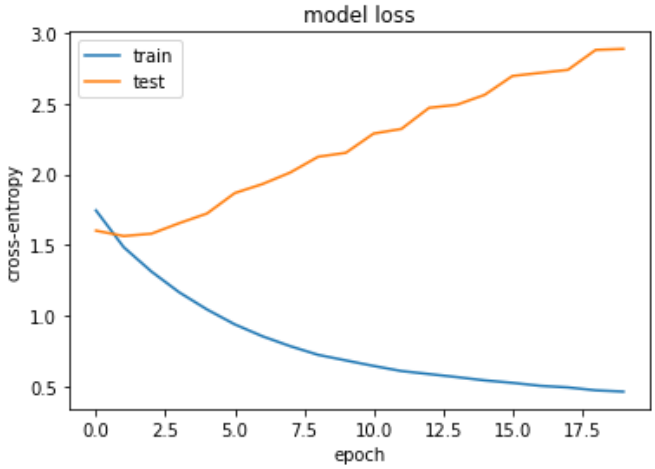

model accuracy
train
test
accuracy
epoch
0.20
0.22
0.24
0.26
0.28
0.30
0.32
0
10
20
30
40
50

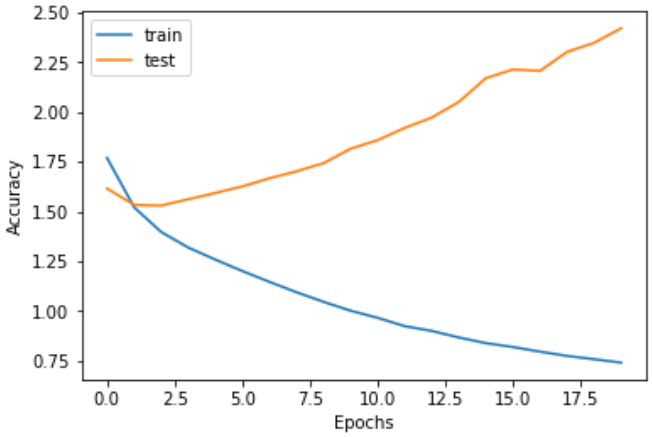

model accuracy
train
test
accuracy
epoch
0.33
0.34
0.35
0.36
0.37
0.38
0.39
0.40
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

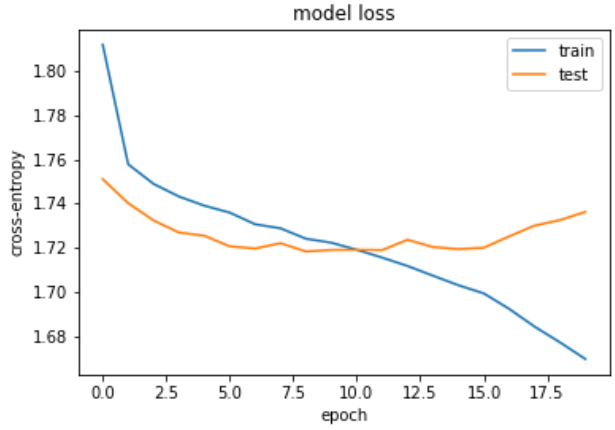


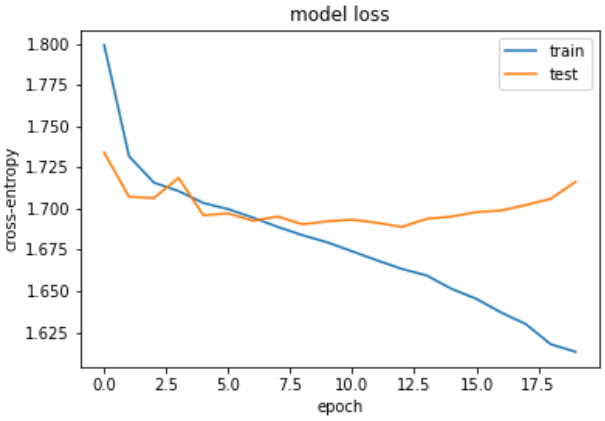


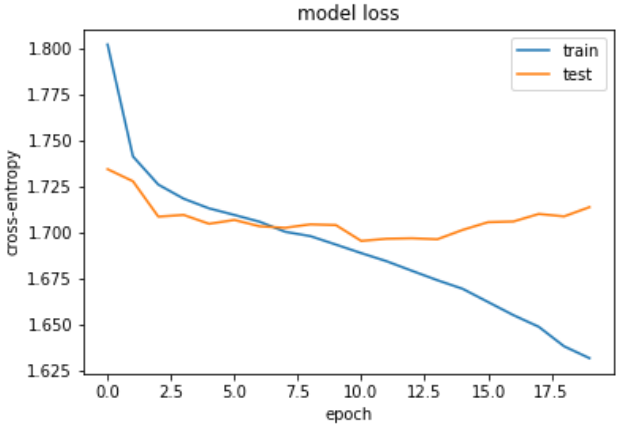
[3] Model Progression – All experiments conducted - Loss

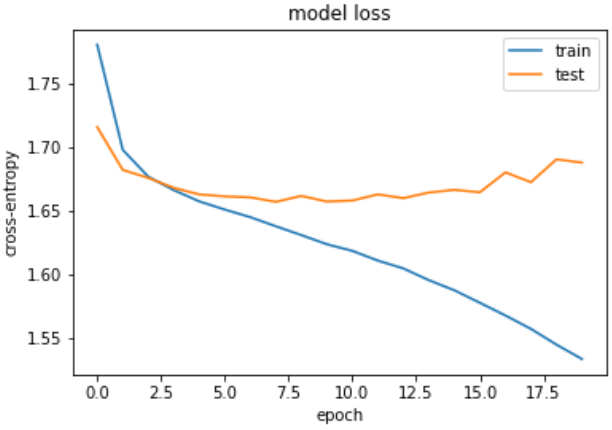


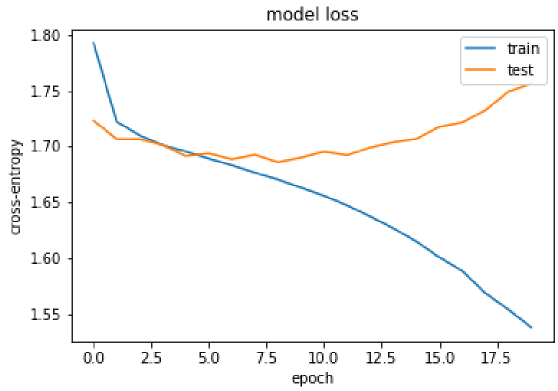


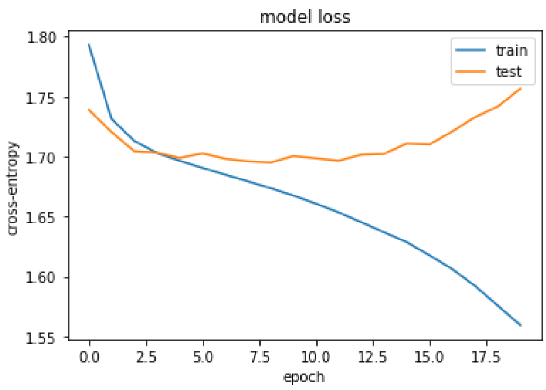


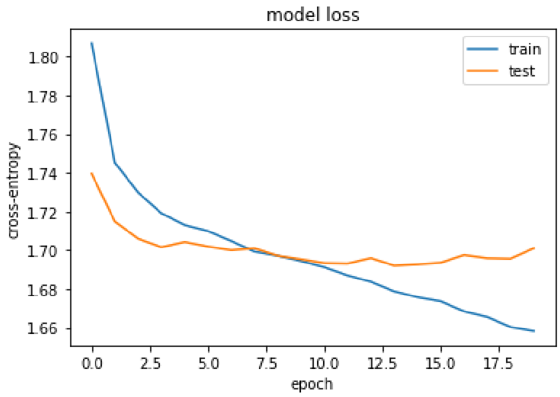


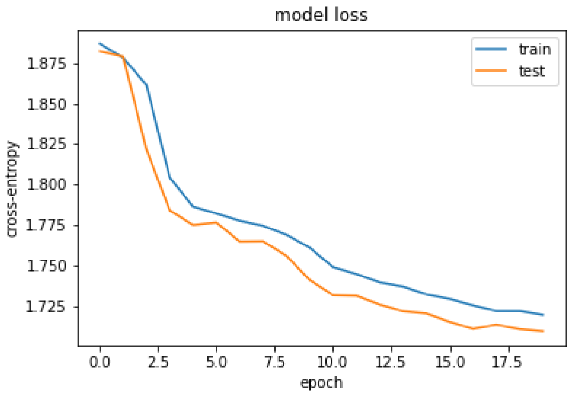


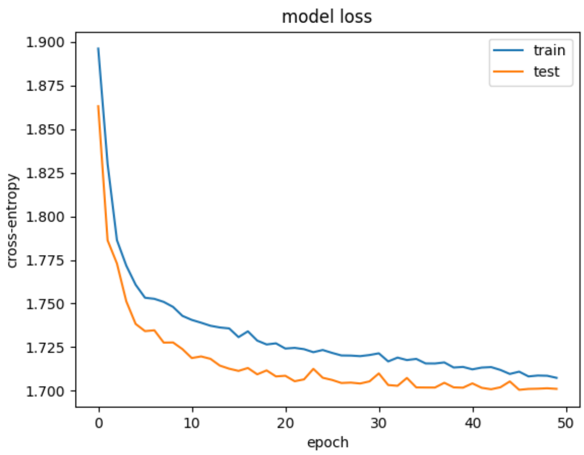


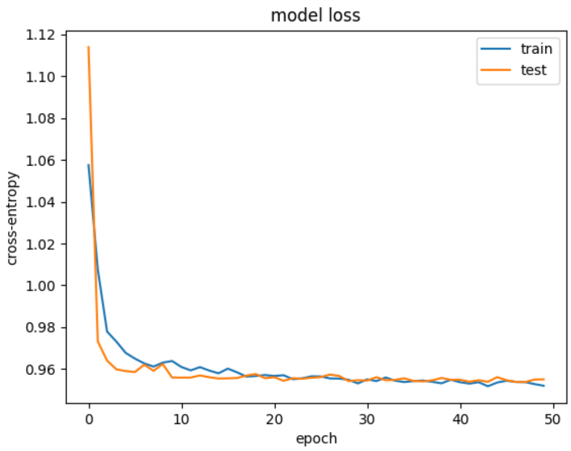

model loss
train
test
cross-entropy
epoch
1.80
1.75
1.70
1.65
1.60
1.55
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


model loss
train
test
cross-entropy
epoch
1.80
1.75
1.70
1.65
1.60
1.55
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


model loss
train
test
cross-entropy
epoch
1.80
1.78
1.76
1.74
1.72
1.70
1.68
1.66
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


model loss
train
test
cross-entropy
epoch
1.875
1.850
1.825
1.800
1.775
1.750
1.725
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


model loss
train
test
cross-entropy
epoch
1.900
1.875
1.850
1.825
1.800
1.775
1.750
1.725
1.700
0
10
20
30
40
50


model loss
train
test
cross-entropy
epoch
1.12
1.10
1.08
1.06
1.04
1.02
1.00
0.98
0.96
0
10
20
30
40
50