Yelp Sentiment Analysis Report

Clyde Wesley Ang	20474922
Hans Krishandi	20373403
Mark Christopher Uy	20466559
tNathaniel Wihardjo	20315011
Petra Gabriela	20315671

1. Introduction

Natural language processing in big data has many useful applications to solve real life problems. One of them is sentiment analysis, which is the process of identifying and categorizing opinions expressed in a piece of text, in order to determine whether the writer's attitude towards a certain topic is positive, negative, or neutral. It is often known as another term called *opinion mining.* In social media, sentiment analysis is extremely useful to perform monitoring and gain overview of the wider public opinion regarding certain topics.

On this occasion, we have the opportunity to take part in a challenge held by Yelp, a social networking site that allows users to post reviews and rate businesses, to conduct research on its dataset and perform sentiment analysis. The task is to classify user reviews in the dataset into ratings from 1 to 5, indicating whether a certain comment is positive or negative. It aims to make use of the data in an innovative way and devise a natural language processing model that has a highly accurate performance.

This report will first discuss the observation of both datasets and different word embeddings matrices. Then, brief information on the environment used and guide to run the model are provided. Different methods, techniques, and architectures will then be further discussed, followed by a detailed explanation of the model that achieved the highest validation accuracy.

2. Data Observation

2.1. Dataset

The data used in this project is downloaded from the Yelp Dataset Challenge[1]. The data consists of reviews of different places by different users. For each data point (review), information about the business, timestamp, user id and business id, full review, and the rank (referred to as 'stars') are extracted. In addition, 'cool', 'useful', 'funny' are provided for each review to indicate the number of votes given by other users.

In terms of the data size, it is observed that a larger training dataset yielded more accurate models. Although 100K data points for the training set is adequate to get more than 60% accuracy, more data should be used in order to achieve higher accuracy and prediction on both test and validation dataset. The proposed model that was able to achieve the highest accuracy uses 300K of training dataset sampled from the full dataset downloaded from Yelp. Our expanded training dataset is carefully chosen so that no data point intersects with the validation and test set.

2.2. Word Embeddings

Word embeddings are used to map words into vectors, which are then used to learn features through additional layers. It allows for dimensionality reduction and contextual similarity, which help in boosting speed and accuracy of the neural network's performance. The model used to obtain word embeddings in this project is called GloVe[2], or Global Vectors, developed by Stanford as an open-source project.

GloVe is a count-based model that learns from co-occurrence information. By training the model on billions of tokens, GloVe produces a word embedding that is capable of capturing the semantic similarity between different words. As most of the words used in the Yelp dataset are common words, the Common Crawl pre-trained word embedding was chosen for this project instead of Wikipedia or Twitter pre-trained embeddings. Additionally, the accuracy of using Common Crawl pre-trained embeddings with 42B tokens (vocabulary) is found to be better than pre-trained embeddings using tokens from Wikipedia and other sources.

FastText[3], which is provided by Facebook Research, also provides pre-trained word embeddings using their own model, which is different from GloVe. It uses tokens collected from two similar sources with GloVe, Common Crawl and Wikipedia, in addition to other sources. Both FastText

[1] https://www.yelp.com/dataset/challenge
[2] https://nlp.stanford.edu/projects/glove/
[3] https://fasttext.cc/docs/en/english-vectors.html

and GloVe provide pre-trained embeddings on different embedding sizes. However, through our experimentation with different model architectures, we found that GloVe pre-trained embeddings resulted in better accuracy than FastText.

3. Training Environment

The primary training environment used for this project was Google Colab[4]. It is a free cloud service that provides both GPU and TPU[5] support, allowing us to train, run, and develop deep learning models in a reasonable amount of time. Most experimentational work was done on this free platform, but in case we needed to run a model that required more memory, the Intel cloud service was also used. However, as the Intel cloud service does not provide a GPU, training time was significantly slower.

3.1. Guidance for Running the Model

There are a few things to be done in order to reproduce our results and run the code provided. One first needs to download the pre-trained Stanford GloVe Embeddings (42B tokens, 300D embeddings). Another requirement is to download the whole Yelp dataset online, which is around 8GB in size, and to perform random sampling to expand the training size to 300K, while ensuring no intersection with the validation and test set. The original extended training dataset used by the authors is also available for download, as specified in the appendix.

All models tested were written with Keras[6], which utilizes TensorFlow[7] as the backend. These are already pre-installed in Google Colab, together with other packages like *NLTK*, *sklearn*, *matplotlib*, and several other basic Python libraries.

TPU hardware acceleration was used for each training. It is observed that TPU provides significantly faster training time for each epoch compared to GPU acceleration. To run the model with GPU instead of TPU, change the layers on line 235 and and 237 from “GRU” layers to “CuDNNGRU”. In addition, to run the model with the absence of TPU, comment out line 259 to 265, and change the variable “tpu_model” to “model” on lines 268, 271, 274, and 278.

Finally, after the adjustments above are done on the code, use the command ‘python asg1.py’ to run the model.

[4] https://colab.research.google.com/
[5] https://en.wikipedia.org/wiki/Tensor_processing_unit
[6] https://keras.io/
[7] https://www.tensorflow.org/

4. Experimentation and Results

4.1. Text Encoding

Different embedding models were experimented with to determine which one was able to yield the highest accuracy. Initially, randomly initialised embedding weights were used for the model and optimised during the training phase. However, it took more time for the model to converge and would lead to overfitting in some cases. This led us to use the pre-trained word embeddings matrix. Different pre-trained word embedding matrices such as GloVe and FastText with different embedding dimension sizes and number of tokens, as well as sentence encoder provided by TensorFlow were experimented with to achieve the best results. Character-level encoding was also tested instead of on the word- or sentence-level encoding.

Universal Sentence Encoder[8] provided by TensorFlow provides a 512 dimension size of sentence vectors, capable of encoding the whole sentence rather than just words. It is expected that the Universal Sentence Encoder is able to capture and understand more meaning, as it learns the whole sentence rather than just the word-level. However, this encoding technique provided no significant increase in the accuracy compared to the word-level encoding with pre-trained embeddings. This encoding method was experimented with BiDirectional LSTM layers and Fully Connected Layers. Considering the time needed to encode each review one-by-one and pre-process the data, word embeddings with pre-trained word embeddings was chosen for its speed.

In addition, character-level encoding[9] was also tested. Character-level encoding provides a more detailed approach that encodes the text on a character-level, rather than on a word-level. This method is proposed as a solution to tackle the disadvantages of absent words in word-level embeddings. As a result, vocabulary present in the test and validation set but absent in the training set would be predicted more accurately. Keras has built-in functions which enable us to do this easily. However, the character-level encoding produces the same accuracy as the randomly initialised word embeddings which is lower than using pre-trained embeddings. Thus, we proceeded with training our model using the pre-trained word embeddings from GloVe.

4.2. Features Extraction

Other available features besides textual content of the review were tested to observe its performance on making the model more general. Some features from the data used are the

[8] https://tfhub.dev/google/universal-sentence-encoder/2
[9] X. Zhang, et.al. (2015). “Character-level Convolutional Networks for Text Classification”. Available: https://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.pdf

'cool', 'funny', and 'useful' features as well as the *subjectivity* and *polarity* of the sentiment analysis provided by TextBlob[10] library.

The values from TextBlob represent whether the whole review is subjective, and whether the review is positive or negative. In addition, average stars and total review count for each business id were extracted from the full Yelp dataset and added to the model as two additional features.

Finally, pycorenlp[11], which provides a Python wrapper for Stanford CoreNLP[12], was utilised to annotate the sentiment analysis values over the textual content of the review, ranging from 1 for negative, to 5 for positive.

These eight additional features were tested on Bidirectional LSTM architecture. The result and discussion of the features extraction is provided on Section 4.3.1.

4.3. Architectures Tested

Different architectures were experimented with to see which model is able to better capture the sentiment of the reviews. All models and different architecture were trained on optimised hyperparameters and optimisers. In order to produce an unbiased result, multiple runs were done on each model and the average results are shown in Table 1. The models were trained with the expanded dataset (300K), unless otherwise specified.

Neural Network Model	**Average Validation Accuracy**
Bidirectional LSTM	62.4%
Bidirectional LSTM + Additional Features	65.8%
CNN - BiLSTM	68.2%
LSTM - CNN	68.8%
Hierarchical Attention Network	69.1%
Time Distributed Bidirectional GRU (discussed on Section 4.3)	70.4%

[10] https://textblob.readthedocs.io/en/dev/
[11] https://github.com/smilli/py-corenlp
[12] https://stanfordnlp.github.io/CoreNLP/

Table 1. Average Validation Accuracy for Different Architectures

4.3.1. Bidirectional LSTM

One of the models that we experimented with is RNN architecture, as it provides the model some memory to better understand the meanings of the text. To tackle the disadvantage of vanilla RNN, we tried LSTM layers to provide more long-term memory to the model. However, to solve the dependency of hidden variables being captured in only one context, bidirectional LSTM is proposed with three fully connected layers. RMSprop is used for the optimizer with learning rate 0.005 to avoid overshooting for 20 epochs. In this model, RMSprop performs better than Adam, as RMSprop works best with the LSTM layers. With GloVe pre-trained embeddings and input length 100, the model was able to achieve a 62.4% validation accuracy.

On another experiment, the additional features discussed on Section 4.2 is appended after the Bi-LSTM layer. Three additional fully connected layers with 100 hidden nodes were used after the addition of the features for the model to optimize the weights and extract important features. The additional features yielded an increase in the validation accuracy of about 3 percent to achieve 65.8%. However, as the pre-processing for original dataset (100K of data points) could take several hours to finish, it became impractical to use for larger datasets. As our final model was able to pass the 70.25% validation accuracy benchmark, these additional features were not used due to its impracticality.

4.3.2. CNN-BiLSTM

Another model tested was the CNN-BiLSTM architecture. This model performed surprisingly well, especially compared to the standard single-layer Bidirectional LSTM models. We pass the embedded input into a 1 dimensional convolution of kernel size 5 with 64 filters in order to learn different features from the sentence structures. A max pooling layer with pool size 3 was then added to generalize the model, and batch normalization was added to reduce overfitting and independently learn features from the LSTM layer.

After getting general features from the CNN layer, we pass the result into a standard Bidirectional LSTM so that the model can learn the connection between these features. We used an LSTM model with 100 hidden units and added dropout with rate 0.5 to prevent overfitting. The LSTM returns the final recurrent block, which is then passed on to the final dense layer for classification.

This model was able to achieve a 68.2% validation accuracy, which is a boost of 2.4 percent. This shows that adding a CNN layer enhances the result and enables the model to predict the classifications more accurately.

4.3.3. LSTM-CNN

Another interesting method we tried was using an LSTM-CNN model. This model was able to slightly outperform the CNN-BiLSTM model. In order to make the dimensions of the parameters match, it was necessary to activate the return sequences parameter for the LSTM layer. By activating this parameter, the LSTM layer returns the hidden state of each recurrent block instead of just the final recurrent block. After this, we reshape the LSTM layer output to a 4 dimensional tensor, which we can perform 2D convolution on.

Convolution was done by using 64 filters with filter sizes of 3, 4, and 5 words. We then do a one layer convolution, followed by a max pooling with a pool size of 2. We concatenate the max pooling layers for each of the filter sizes, and then perform dropout. The features are then passed to one more dense layer for prediction and classification.

By adding a CNN structure before the dense layer, we are able get a better and more informative set of features in contrast to using only the last hidden block output of the LSTM layer. This is why the dense layer at the end overfits less on the original dataset, and performs better during validation.

4.3.4. Hierarchical Attention Network

Hierarchical Attention Network, which was proposed by a group of researchers from Carnegie Mellon University and Microsoft, was also tested for this classification problem. The main idea behind this architecture is that texts consist of sentences and sentences consist of words. The architecture works by feeding a recurrent network layer with the words in the text and time distribute the words in each sentence to an attention network. The output of this time distributed attention network is again fed to a recurrent network layer which acts as the sentence encoder, followed by another attention network that serves as the sentence attention.

Compared to previous models, this architecture performs quite well with a 1 percent boost in validation accuracy. Each input data is transformed into a 2D matrix, with 10 rows (10 sentences) and 10 columns (10 words per sentence). We also tried using 12 sentences and 15 words per sentence. However, input data with shape larger than 10x10 matrix does not give a noticeable increase in validation accuracy but adds a considerable amount of training duration. Using RMSprop optimizer with 0.001 learning rate and GloVe word embeddings, this model achieved 69.1% validation accuracy.

Considering the long training time (about 2.5 hours training period on Intel AI DevCloud) and insufficient validation accuracy, we decided to devise another deep learning architecture, which produced the best accuracy with practical training time, as described in the next section.

4.4. Final Model and Hyperparameters

After experimenting with different deep learning architectures discussed on Section 4.3, we decided to use the following model due to its performance in terms of accuracy and training duration. In its core, this model consists of Embedding, Bidirectional GRU, and Time Distributed layers. The diagram of this deep learning architecture is shown in Figure 1.

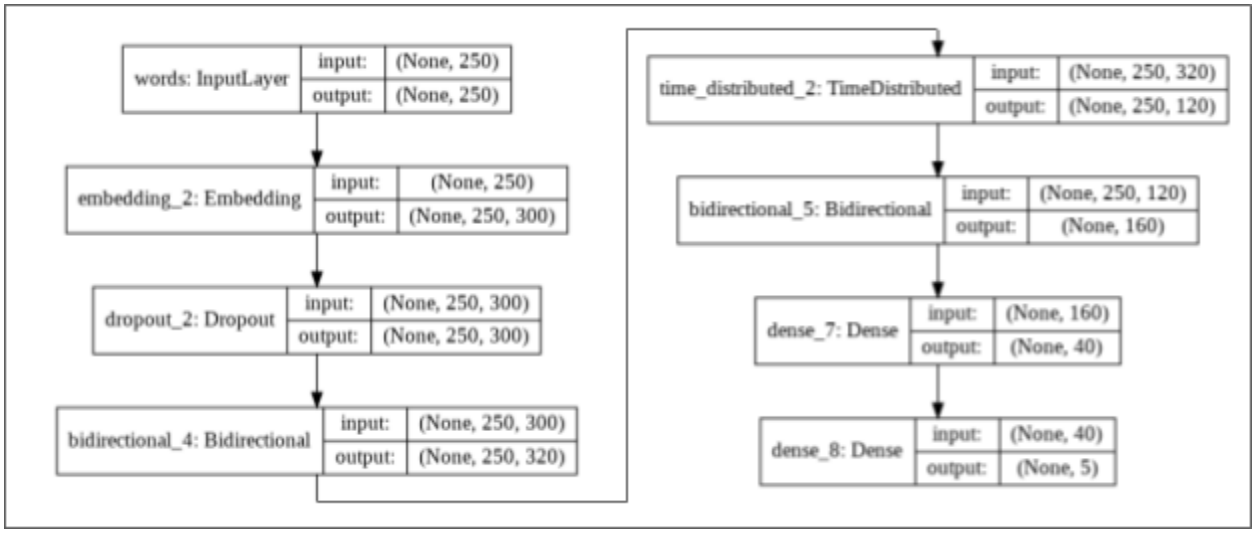

Figure 1. Diagram of Final Model Architecture

First, each review data is transformed into a tensor of 250 elements. These 250 elements represent the first 250 words of the review text with each word converted into the index of that word in the vocabulary dictionary. We chose 250 as the sequence length because the statistics of the review data shows that the maximum number of words per review text is about 250 words. Therefore, by using 250 as the sequence length, it is expected that this model can learn and predict more accurately based on the full text review.

This tensor is then fed to the embedding layer whose weights are set with the pre-trained GloVe word embeddings. These weights are set to be non-trainable to avoid overfitting phenomenon. This embedding layer will then output a three-dimensional matrix, representing the number of samples, the input length or number of words, and the word vectors. To further reduce the possibility of overfitting, a dropout layer with 0.2 dropout rate was added after the embedding layer.

The next layer is a Bidirectional GRU with 160 hidden units and 0.2 dropout rate. An L2 kernel regularizer with 1e-5 penalty was added to this layer to help the model learn from the data more generally. In this architecture, a bidirectional layer is used to handle the one-sided context problem faced by unidirectional RNNs. By using the bidirectional recurrent layer, this model is able to learn the context of the text better. In addition, we decided to use GRU instead of LSTM because GRUs train faster than LSTMs in general[13]. Even though LSTMs are able to memorize longer sequences[14], in this classification problem, words in different sentences are not highly related to each other. Thus, GRUs are more efficient in this classification problem. With the configuration for the model to return the hidden states for each timestep, the output of this layer is a matrix with 250 rows and 320 columns.

Using a Time Distributed layer wrapping a 120-neuron dense layer, the output of each time step is passed individually. By focusing on each word individually, the model is able to have a deeper knowledge about each word while ignoring unnecessary connections between words at the same time.

The model then learns the connections between the most important words in the following GRU layer. A Bidirectional GRU layer with 80 hidden units, 0.2 dropout rate, and 1e-5 penalty L2 kernel regularizer computes the dependencies between words and outputs the hidden state at the last time step.

The last two layers are the plain fully-connected layers with 40 and 5 units respectively. The last dense layer with 5 units uses a softmax activation function since this is a classification problem. The output of this layer, which is also the final output of this model, will be compared with the target label that has been transformed into a five-element vector using One-Hot Encoding.

The number of hidden units in each layer is set to be gradually decreasing so that there is less information loss during forward computations. Using RMSprop optimizer with 0.001 learning rate and automatic learning rate reduction, this model is able to achieve 70.4% validation accuracy. The model accuracy and loss plot is shown in Figure 2.

[13] W. Yin, et.al. (February, 2017). “Comparative Study of CNN and RNN for Natural Language Processing”. Available: https://arxiv.org/pdf/1702.01923.pdf
[14] J. Chung, et.al. (December, 2014). “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”. Available: https://arxiv.org/pdf/1412.3555v1.pdf

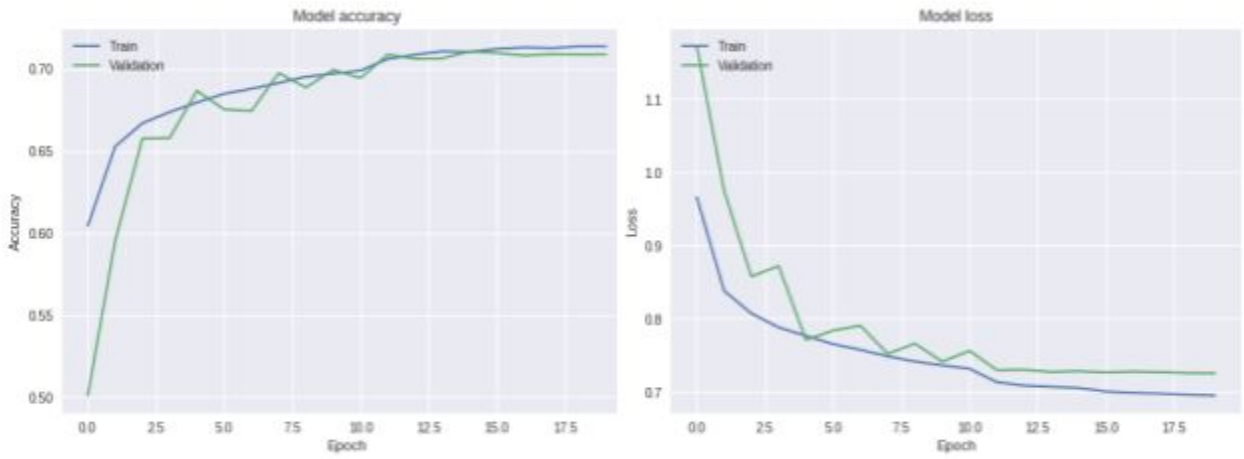

Figure 2. Accuracy and Loss against Number of Epochs of the Final Model

5. Conclusion

Yelp Dataset challenge presents the opportunity to gain more insight into the application of sentiment analysis and experience the process of designing a suitable deep learning model to get accurate results. Different techniques of encoding and features extraction, as well as different architecture approaches have been experimented with to evaluate the performance on Yelp dataset. In the end, a combination of Embeddings, Bidirectional GRU, and Time Distributed layers were observed to yield the best results, reaching 70.4% validation accuracy on the sentiment analysis task with practical training time on Google Colab. All in all, we hope that this research could be beneficial for Yelp and give better automation on inferring ratings from user reviews.

Appendix

Original extended training dataset used by the authors is available in the link below:
https://drive.google.com/open?id=1vpkQdRRptnF4BpFPEMKs1j-fJKGINZcv