Yelp Me Rate This: Using Deep Learning to Predict Yelp Review Ratings

Aatif Jiwani* Eshaan Pathak* Varun Wadhwa*
{aatifjiwani,eshaanpathak,vwadhwa88}@berkeley.edu
University of California, Berkeley

1. Introduction

Yelp is a company that hosts reviews on their website, www.yelp.com, where users can review and recommend businesses to other interested customers. Reviews are helpful for businesses and customers as they can provide advice to businesses on how they can improve their services and inform other potential customers on the services they should expect to receive at these businesses.

The dataset used in this project is a public dataset that is a subset of the Yelp Review Dataset released by Yelp and is available at www.yelp.com/dataset. Each datapoint consists of a review ID, review text, and review rating out of 5 stars. The dataset primarily contains reviews of businesses, such as restaurants, hair salons, venues, doctors, etc., that provide services to customers. Customers review a business regarding the business's quality of service and treatment of customers.

Students wanting to learn more about applying natural language processing (NLP) techniques as well as NLP researchers care about this problem. Through analyzing reviews, students and researchers can quantify how helpful reviews are and how reviews can correlate to the success of a business.

The objective of this project is to predict the rating of a review given the review. In order to predict the rating of a review, machine learning models must encapsulate features that can assist them in predicting the most likely rating. With these features, the commonalities between reviews with the same rating and differences between reviews with different ratings can be explored. In this work, we explore deep learning architectures[1], such as bidirectional LSTMs and Transformer Encoders, and more state-of-the-art models like DistilRoBERTa, RoBERTa, and XLNet.

2. Data Preparation

2.1 Data Cleaning and Vocabulary Generation

The raw reviews in the Yelp training dataset have a high variance in terms of how they are written, as people have different preferences of contraction usage, symbol usage, date and time formatting, and even spelling of slang words. By treating a sequence of alphanumeric characters separated by a space as its own word, there are approximately 530,000 unique tokens in the raw training dataset without any cleaning or preprocessing. Clearly, this naive algorithm wouldn't suffice to create a vocabulary because not only is it too large, but there are many semantic duplicates. For instance, people tend to write either "don't" or "do not", some type numerals or the number words ("5" vs. "five"), and many reviewers elongate their punctuation marks ("!!!!!").

Also, having a very large vocabulary is not feasible because an embedding of size 200 would create over 100 million parameters for the embedding matrix. Therefore, we sought to reduce this number in order to create a reasonably sized vocabulary of around 50,000 words. In order to make this possible, we employed regular expressions (regex) to clean each review using the following rules:

1. Miscellaneous symbols, like **parentheses**, **brackets**, and **extra punctuation marks**, are removed.
   "Food was bad (it was cheap, though)" ⇒ *"Food was bad it was cheap, though"*
2. **URLs** are removed
   "Refer to my site www.yelp.com" ⇒ *"Refer to my site"*
3. Accepted symbols are separated to **include a space** between each symbol
   "This restaurant was so good!!!" ⇒ *"This restaurant was so good ! ! !"*
4. **Contractions** are expanded into their full form
   "I wouldn't recommend this" ⇒ *"I would not recommend this"*
5. **Numbers** are rewritten as words
   "I came on 1-1-20" ⇒ *"I came on one - one - two zero"*

---

*Equal contribution. Refer to **Section 7**.

[1]Code repository can be publicly accessed via https://github.com/aatifjiwani/yelp-rating-predictor. Follow the README for instructions.

Figure 1: Illustration of regular expression cleaner used on a sample review that violates multiple rules

Refer to **Figure 1** for an example of combining multiple rules together. After cleaning up the reviews, we also used `nltk` to stem each cleaned review ("therapist" to "therap"), so some words, such as "therapist" and "therapy," can be viewed as similar features due to their identical root stems. Cleaning and stemming each word reduced the training dataset to approximately 100,000 unique tokens. To create the vocabulary, we simply used a counter and saved and indexed the top 50,000 stemmed and cleaned tokens. The vocabulary includes special tokens "<PAD>" and "<UNK>" which are indexed as 0 and 1, respectively. To our surprise, the least-occuring words in our 50,000 token vocabulary contained misspelled words. This must mean that some words are just commonly misspelled and this may play a role in prediction power.

2.2 Word-Piece Tokenization

Once the vocabulary is generated, tokenizing reviews to be tossed into a model is fairly straightforward. Instead of tokenizing the reviews all at once, we perform tokenization dynamically when the respective input is requested for training. The raw review, identified by a unique ID, is taken from the dataset, cleaned, stemmed, and tokenized into its separate words or symbols. Each element is then converted into its respective vocabulary index. Refer to **Figure 2** for an illustration of what tokenization and stemming looks like on a cleaned review. Notice how the misspelled word "*jibberish*" and "*abcd*" are converted to "<UNK>" tokens because both words do not exist in the vocabulary of 50,000.

2.3 Byte-Level BPE Tokenization

Byte Pair Encodings (BPE) have become a popular technique in NLP, notably being used with Transformer architectures. BPE is a top-down compression technique that replaces frequent pairs of characters, or pairs of bytes, with another character or byte that does not exist within the text corpus. This is repeatedly done in a recursive manner until the vocabulary limit has been reached. We used huggingface's tokenizers package for Python and trained a Byte-level BPE Tokenizer with a vocabulary size of 25,000. The reason for the lower vocabulary size, compared to the 50,000 detailed above, is that we wanted an abundance of prefixes and suffixes in our vocabulary (i.e. "ization") so that we wouldn't have to use a stemmer. This would also allow complex words to be separated into common character pairs and would remove the necessity for an unknown token. We also noticed that if we increased the vocabulary size to 40,000 or above, the generated vocabulary would contain misspelled words as Yelp reviews are commonly known for poor spelling.Refer to **Figure 3** for an illustration of what BPE tokenization looks like on the cleaned review. Notice that compared to the base tokenization in **Section 2.2**, the word "*jibberish*" and "*abcd*" are not replaced by "<UNK>" tokens, and instead are separated into the most common byte-pairs that makeup the entire word.

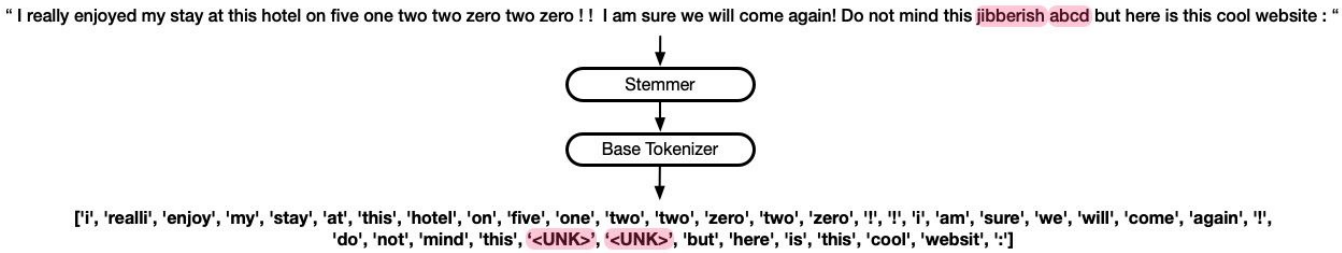

Figure 2: Illustration of stemming and the base tokenizer being used on the cleaned review from Figure 1

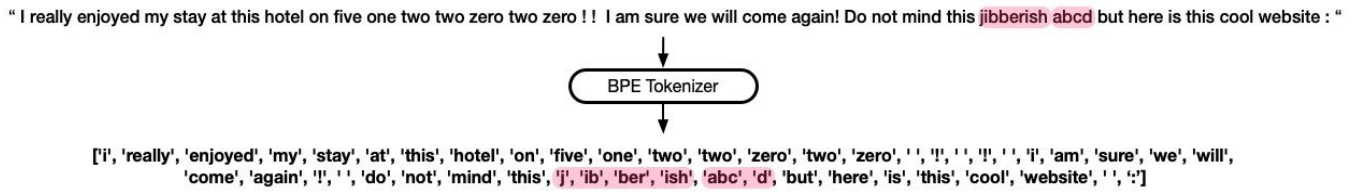

Figure 3: Illustration of Byte-level BPE Tokenizer being used on the cleaned review from Figure 1

This puts BPE tokenization at an advantage because unknown words are no longer replaced by a special token, but are instead broken up into several character pairs that could provide contextual or semantic meaning within the embeddings.

2.4 Pre-Trained Word Embeddings

2.4.1 GloVe Twitter Embeddings

Using the provided GloVe 200-dimensional Twitter embeddings, under the assumption that Twitter and Yelp have similar wordings, all the words in every review were either assigned an embedding from the GloVe Twitter vocabulary or were set as a <UNK> token. Unfortunately, there are quite a lot of words (~33%) in the Yelp dataset that do not exist in the GloVe Twitter vocabulary.

2.4.2 word2vec Embeddings

Word2vec is a popular technique to learn word embeddings by predicting similarities between words. Skip-gram is a training algorithm for Word2vec that predicts target contexts by using input words. Gensim is a library that provides a word2vec model that has already been trained on large corpuses (e.g. Wikipedia, Twitter, etc.). Training this gensim Word2vec model using skip-gram with a window of 3 on all words in the Yelp training dataset produces 200-dimensional embeddings that represent more specifically Yelp words in similar contexts. However, some opposing words that can be used in similar contexts (e.g. “good” and “bad”) have similar embeddings. When predicting Yelp ratings, these pairs of words can actually hurt the performance of the models.

2.4.3 Sentiment-Preserving Embeddings

As mentioned in **Section 2.4.2**, words with opposite sentiments tend to appear in similar contexts, making it difficult for the LSTM model from **Section 3.1** to classify a review. As an attempt to alleviate this, we used sentiment-preserving embeddings, which are modified embeddings that make words with similar sentiments closer to each other (Yu et al. 2017). Embeddings were modified using a lexicon with valence values from 1 to 9 where the valence values range from negative to positive sentiment for each word in the lexicon, respectively. Unfortunately, sentiment-preserving embeddings actually ruined our model’s performance because only a small portion of tokens from the given lexicon appeared in our vocabulary.

We compared the performances between using an unstemmed and stemmed vocabulary with the unstemmed valence lexicon. The unstemmed vocabulary bidirectional LSTM model has a validation accuracy of 74.33%, and the stemmed vocabulary bidirectional LSTM model has a validation accuracy of 76.41%, which are lower than the bidirectional LSTM model without sentiment-preserving embeddings’ validation accuracy of 79.60%. The training and validation splits were 80% and 20% of the full dataset, respectively.

2.4.4 ELMo Embeddings

ELMo embeddings are powerful at representing the syntax and semantics of words, as well as how their uses vary across linguistic contexts. These embeddings are learned from the internal state of a bidirectional LSTM to represent contextual features of the input Yelp reviews. TensorFlow Hub provides ELMo embeddings that have already been trained on large text corpuses and can be fine-tuned using the Yelp training dataset to produce sound Yelp word embeddings. These embeddings outperformed both the GloVe Twitter and word2vec embeddings.

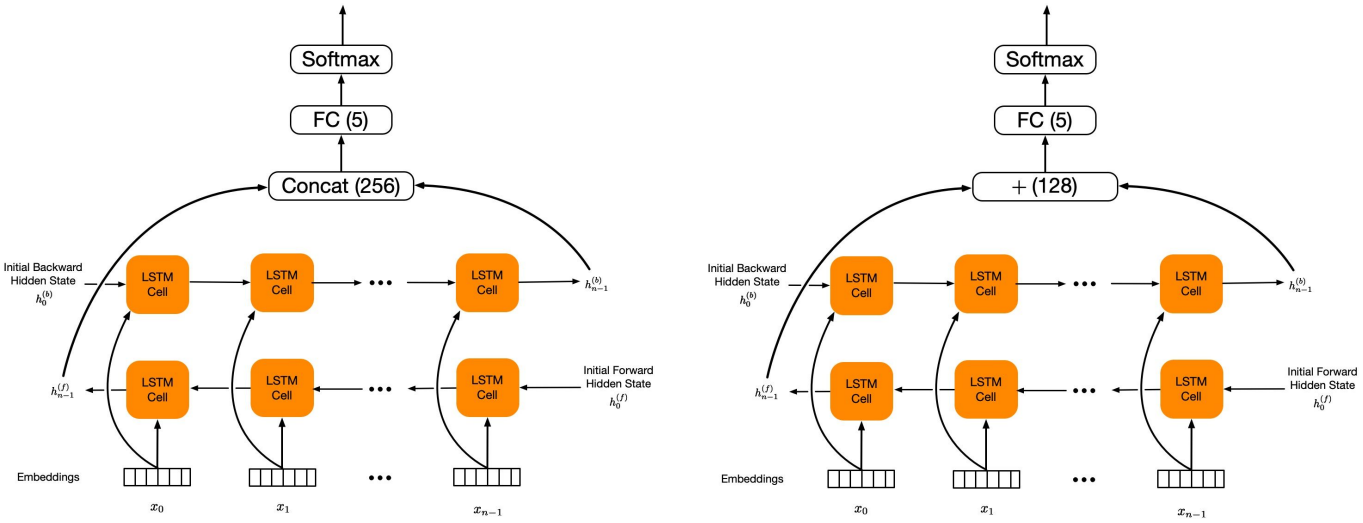

Figure 4: Shown on the *left* is *BiLSTM-Concat* and shown on the *right* is *BiLSTM-Sum*

3. Models

3.1 Bidirectional LSTM

For our baseline models, we employed two bidirectional LSTM networks, *BiLSTM-Concat* and *BiLSTM-Sum*. Both models have the same base encoder but both differ in the decoder module of the architecture. The encoder in both of the models is a 1-layer bidirectional LSTM with a model dimension of 128. We also use the pre-trained ELMo embedding matrix from **Section 2.4.3** with no fine-tuning (gradient calculations and updates are set to False). The input to the LSTM encoder is a design matrix $X \in \mathbb{R}^{B \, x \, S}$ where B is the batch size and S is the sequence length. For both LSTMs, we use a max sequence length of 1000 and use padding on the end of the sequence that falls short. The output from the LSTM that we use is the final hidden states of the encoder which comes in the form of a matrix $H \in \mathbb{R}^{2 \, x \, B \, x \, 128}$ where $H[0]$ is the final forward hidden state and $H[1]$ is the final backward hidden state. *BiLSTM-Concat* concatenates $H[0]$ and $H[1]$ into a 256-dimensional matrix and feeds to the fully connected (FC) network followed by a softmax to yield predictions. On the other hand, *BiLSTM-Sum* adds $H[0]$ and $H[1]$ into a 128-dimensional matrix and feeds it into a smaller FC network but similarly followed by a softmax. Refer to **Figure 4** for an illustrative diagram of the differences between both bidirectional LSTMs.

3.2 Transformer

Transformers gained popularity from the *Attention is all you need* paper (Vaswani et al. 2017) and have been used for natural language understanding (NLU) and natural language generation (NLG) tasks. This problem of predicting Yelp star ratings given a review is along the lines of NLU, and therefore we only employ transformer encoders followed by a FC-net and softmax decoder for this classification task. In this project, we created and experimented with 5 different variations of transformer encoders. In each of the following transformers, we used 4 heads, a 512-dimensional feed-forward network, and a cross-entropy loss. The output of each transformer is a matrix $Z \in \mathbb{R}^{BxSxD}$ where D is the dimension of the transformer. All models weights were initialized using Depth-Scaled Initialization (Zhang et al. 2019).

Transformer-256 used a 256-dimensional Transformer Encoder with 4 layers, used a dropout of 0.2, and accepted an input with max-length of 1000. Prediction was done by averaging Z over the 2nd dimension and feeding it to the decoder. The input is tokenized using the base tokenizer in Section **2.2**.

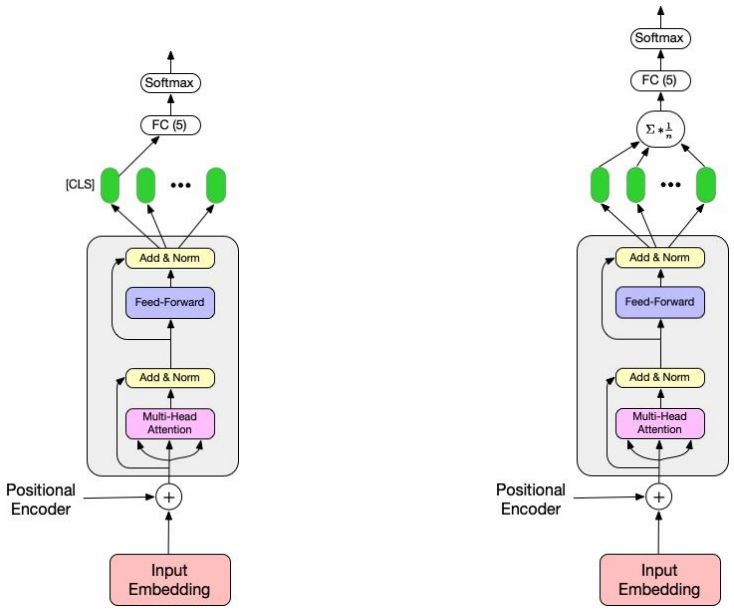

Figure 5: Shown on the *left* is a Transformer architecture that picks out the <CLS> token output to insert into the decoder and shown on the *right* is a Transformer architecture that averages over the output's sequence dimension.

Transformer-256-CLS used the same architecture as *Transformer-256* but the input included a "<CLS>" special token at the beginning of the input and used the corresponding output position in Z to feed to the decoder.
Transformer-360-WCE-BPE used a 360-dimensional Transformer Encoder with 4 layers, a dropout of 0.2, but accepted an input with max-length 250 and tokenized via the Byte-Level BPE Tokenizer. Also, the cross-entropy loss was weighted using the following weights: $[\ 0.8,\ 1.4,\ 1.4,\ 1.2,\ 0.8\ ]$, each corresponding to the respective star rating.
Transformer-512-WCE-BPE used a 512-dimensional Transformer Encoder, but otherwise has the same parameters and specifications as *Transformer-360-WCE-BPE*.
Transformer-5Layer-WCE-BPE used a 5-layer 256-dimensional Transformer Encoder, but otherwise has the same parameters and specifications as *Transformer-360-WCE-BPE*.

We employed a weighted cross-entropy (WCE) because 1 and 5 star ratings are the most abundant within the dataset. Therefore, we want to penalize the model more for misclassifying star ratings 2, 3, and 4. Star ratings 2 and 3 appear the least so they are weighted the highest. A 4 star rating is the next least abundant, so it has a slightly lower weight. A 1 and 5 star rating appear the most often so they have weights less than 1 to not penalize the model as much for misclassifying those reviews. Refer to **Figure 5** for an illustrative difference between the Transformer models that averages Z over the sequence dimension and takes the corresponding output of the "<CLS>" special token.

3.3 Pre-Trained Transformer Models

Each of the below models were trained for one epoch using an Adam optimizer with a learning rate of 4e-5 and Adam epsilon of 1e-8. Due to time constraints, the optimizer, learning rate, and number of epochs were kept at these defaults. Each model had its own embedding layer (not from **Section 2.4**).

These models were also all trained on the entire Yelp training dataset without splitting into a training and validation dataset. Instead, every 5000 iterations in the epoch, a model checkpoint was created. These checkpoints mimic different splits in the dataset (i.e. a checkpoint at batch 25000/33349 represents a 75% split between training and validation, whereas a checkpoint at batch 30000/33349 represents a 90% split between training and validation). The checkpoint with the lowest mean average error and highest accuracy on the released challenge datasets was then chosen to minimize underfitting/overfitting.

DistilRoBERTa

DistilRoBERTa is an equivalent model to RoBERTa except that it uses half the parameters. Due to memory and time constraints, instead of using RoBERTa, DistilRoBERTa was used to more efficiently provide insight into changing hyperparameters and testing whether the cleaned data actually produced better results than the raw data. The base model was chosen: 6 layers, 768 hidden states, 12 heads, 82M parameters.

The first version of fine-tuning used a max sequence length of 256 and batch size of 32 with cleaned Yelp training data. The second version of fine-tuning used a max sequence length of 256 and batch size of 32 with raw Yelp training data. The third version of fine-tuning used a max sequence length of 128 and batch size of 32 with cleaned Yelp training data.

RoBERTa

RoBERTa is a bidirectional transformer that is pre-trained over textual data to learn a language, which is then fine-tuned to predict Yelp ratings. It is an optimized BERT approach that has been pre-trained using dynamic masking instead of next sentence prediction. The base cased model was chosen due to memory constraints: 12 layer, 768 hidden states, 12 heads, 125M parameters.

The first and second versions of fine-tuning used max sequence lengths of 256 and 128, respectively, with batch sizes of 16 for both with cleaned Yelp training data.

XLNet

XLNet is a large bidirectional transformer that improves BERT's training performance by introducing permutation language modeling: all tokens are predicted in random order. This helps the model learn bidirectional relationships, which allows it to better handle dependencies/relations between words. The base cased model was chosen due to memory constraints: 12 layers, 1024 hidden states, 16 heads, 110M parameters.

The first version of fine-tuning used a max sequence length of 512 and batch size of 8 with cleaned Yelp training data. The second version of fine-tuning used a max sequence length of 128 and batch size of 16 with cleaned Yelp training data.

4. Experiments and Results

4.1 Bidirectional LSTM and Transformer Encoders

Before training, we split the training dataset into train and validation sets using an 80% and 20% split, respectively. Also, all models used early stopping with a patience of 2 to prevent the model being saved when it is starting to overfit the training data.

Both baseline LSTM models were trained with a batch size of 64 and for 10 epochs. We used an Adam optimizer with the initial learning rate of 0.001. Refer to **Figure 6** for a plot of validation accuracies on the non-challenge training dataset. All transformer models were trained with a batch size of 32 and for 10 epochs. Some models however, as evident in **Figure 6**, were stopped early due to the validation loss not improving in 2 epochs. The transformer models were also trained using SGD but with different learning rate schedules as detailed below:

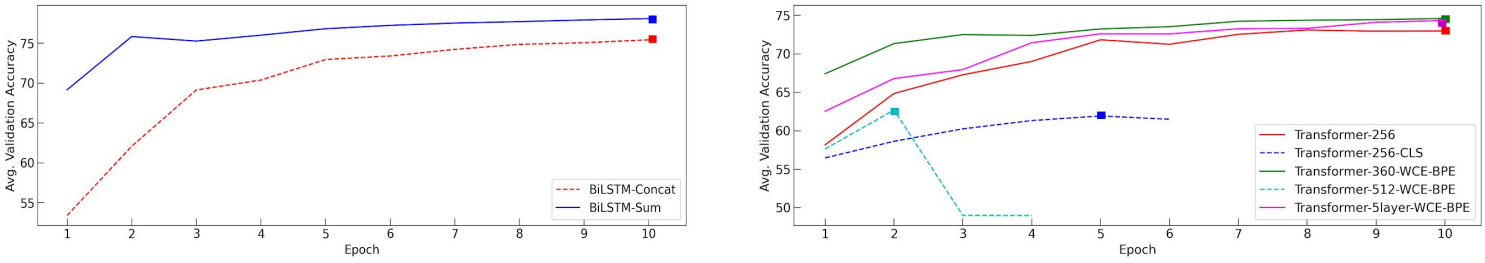

Figure 6: Shown on the *left* is a validation accuracy plot for the BiLSTM models and shown on the *right* is a validation accuracy plot for the Transformer models

Transformer-256, Transformer-256-CLS, and *Transformer-360-WCE-BPE* were trained using a step learning rate scheduler that multiplied the current learning rate by a factor of 0.85 every epoch. These models used a max length of 250.

The *last two transformer models* were trained using a warmup learning rate scheduler (WLRS) (Popel et. al 2018) with 30,000 steps, an end learning rate of 0.2, and an initial learning rate of 0.07. These models used a max length of 500.

	MAE Set 3	**Acc. Set 3**	**MAE Set 5**	**Acc. Set 5**	**MAE Set 6**	**Acc. Set 6**
BiLSTM-Concat	0.601	0.504	0.884	0.216	2.238	**0.426**
BiLSTM-Sum	**0.483**	**0.578**	**0.69**	**0.39**	**2.228**	0.416
Transformer-360-WCE-BPE	0.646	0.502	0.956	0.22	2.44	0.372
Transformer-5L-WCE-BPE	0.655	0.502	1.026	0.182	2.43	0.38

4.2 Pre-Trained Transformer Models

The best model checkpoint was consistently around the 90% split mark (e.g. at batch 30000/33349 with a batch size of 16). Training over 100% of the dataset clearly overfitted the model, while training over less than 90% produced underfitting. Here are the mean absolute error (MAE) and accuracy results for each version of each model at the 90% split mark on the fifth and sixth challenge datasets:

	MAE Set 5	**Accuracy Set 5**	**MAE Set 6**	**Accuracy Set 6**
DistilRoBERTa V1	0.628	0.412	2.258	0.388
DistilRoBERTa V2	0.636	0.408	2.276	0.384
DistilRoBERTa V3	0.590	0.454	2.204	0.402
RoBERTa V1	0.684	0.372	2.068	0.404
RoBERTa V2	0.636	0.430	1.872	0.452
XLNet V1	0.728	0.368	2.108	0.418
XLNet V2	**0.684**	**0.380**	**1.874**	**0.472**

DistilRoBERTa V3 performed best on Challenge Set 5, but there is little variation among the model results for this challenge set. XLNet V2 (with RoBERTa V2 as a close second) performed best on Challenge Set 6. Since there is higher variance among the model results, Challenge Set 6 is determined to be the more accurate threshold for deciding the best model. Thus, XLNet V2 is considered to be the best model for predicting Yelp ratings.

Smaller sequence lengths perform much better than longer sequence lengths. This may be due to how the beginning of a review may be more telling of its rating as the later parts of the review are more of an explanation of the customer's experience with the business. For example, see the raw review below that gave a business a 1 star rating. It has about 680 tokens in its raw format, of which most are not shown for the sake of brevity. Evidently, the review begins with advising the reader to *"stay away from any free deal these people offer for you*" and then proceeds to explain the business's poor service.

> *"Whatever you do, stay away from any free deal these people offer for you to sit through their timeshare presentation!!! On the paperwork it says the presentation lasts for 2-2 and a half hrs. Well when we showed up, our salesman told us we would be with him for the next 3 hrs.* […] *We all agreed it was not worth the free stuff they promise."*

Maximizing batch size with these smaller sequence lengths sped up training and also produced superior results. Too large of a batch size was not a problem due to memory constraints that forced an upper bound possible batch size. For RoBERTa and XLNet, available memory capped the batch size at 16. For DistilRoBERTa, available memory capped the batch size at 32.

Testing cleaned versus raw data on DistilRoBERTa with the same hyperparameters showed minor improvements with cleaned data. Thus, only cleaned data was used for RoBERTa and XLNet.

5. Tools

The models were trained using a virtual machine with a single NVIDIA Tesla K80 GPU through Google Cloud. This GPU provided 11GB of memory, which allowed much larger models (RoBERTa and XLNet) to train without running into memory errors. Utilizing CUDA, a parallel computing platform and API model created by NVIDIA, enabled a 5x to 12x performance improvement on training runtime.

The simpletransformers library uses HuggingFace and PyTorch to provide pre-trained models that can be implemented in just a few lines of code yet can still be customized with arguments. Fine-tuning pre-trained DistilRoBERTa, RoBERTa, and XLNet models with the Yelp training dataset saves an enormous amount of time while still producing high accuracies.

The bidirectional LSTM was implemented first using TensorFlow, but the runtime was much longer than expected. As a result, another bidirectional LSTM was trained using PyTorch, which trained much faster and produced comparable accuracies and errors.

6. Lessons Learned

6.1 Transformers

Based on the results in **Section 4**, the Transformer models surprisingly did not perform as well as the baseline LSTMs. We believe that this is because the reviews have a high variance of word usage, causing difficulties in pinpointing attention, or that the embeddings were not strong enough to make an accurate prediction.

It is interesting to note that in **Figure 6**, using the "<CLS>" special token decreased performance by almost 10%. The reason behind this could be that the special token is mainly used to pre-train larger Transformers, like BERT and RoBERTa, on more standardized datasets. Therefore, the special token is necessary for fine-tuning. However, if we use the special token without any form of pre-training on a less standardized dataset, it seems that it hinders performance since the corresponding output feature is not trained to hold enough information about the entire sequence.

In our experiments we saw that a large Transformer, such as *Transformer-512-WCE-BPE*, diverged within 2 epochs and failed to achieve a better minimum. This model was trained using the WLRS, as specified in **Section 4.1**. In the *Training tips for the transformer model* paper, the authors noted that the Transformer diverged if the warmup steps were too small. This prompts further research in terms of testing different warmup steps for larger transformers.

6.2 Pre-Trained Transformer Models

Overall, XLNet V2 is the best model out of all of the models implemented in this project. It has been shown that XLNet outperforms RoBERTa and DistilRoBERTa when the data is significantly different from the pre-training data, which is true for the Yelp training dataset (**Section 2.4.1** explains that 33% of Yelp words was not found in large corpuses).

Models with smaller sequence lengths performed better than models with larger sequence lengths, which is surprising because it means that the first part of a review gives the model the most relevant information than the entirety of a review. And using cleaned data instead of raw data as input evidently does improve the model results.

6.3 Future Exploration

Due to time and memory constraints, our models could not train on the full Yelp Review Dataset. As a result, we are interested in incorporating distributed training of GPUs to compare the performances of our models. Another avenue to try would be to add more noisy data to make the language models more robust to misspelled tokens and perturbed inputs. More research must be done on the types of noisy datasets to use as data that is as similar as possible to the domain of business reviews is very necessary. Lastly, looking for the specific features as to why a model predicted a rating for a review is another potential endeavor.

7. Team Contributions

Each member equally contributed to this project and also assisted the other two on their main tasks. Aatif worked on building the bidirectional LSTM with PyTorch and Transformer Encoder models. Eshaan worked on the word embeddings for the LSTM, such as training new word embeddings and experimenting with sentiment-preserving word embeddings. Varun worked on implementing the GloVe Twitter embeddings, the word2vec embeddings, the ELMo embeddings, the bidirectional LSTM with TensorFlow, DistilRoBERTa, RoBERTa, and XLNet.

References

Suleiman Khan. BERT, RoBERTa, DistilBERT, XLNet — which one to use?. Towards Data Science.

Martin Popel and Ondřej Bojar. Training tips for the transformer model. arXiv Preprint:1804.00247, 2018.

Rico Sennrich, Barry Haddow and Alexandra Birch. Neural machine translation of rare words with subword units. Proceedings of the Association for Computational Linguistics, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. Attention is all you need. Proceedings of the Neural Information Processing Systems, 2017.

Liang-Chih Yu, Jin Wang, K. Robert Lai, and Xuejie Zhang. Refining word embeddings for sentiment analysis. Proceedings of the Empirical Methods in Natural Language Processing, 2017.

Biao Zhang, Ivan Titov, and Rico Sennrich. Improving deep transformer with depth-scaled initialization and merged attention. Proceedings of the Empirical Methods in Natural Language Processing and the Joint Conference on Natural Language Processing, 2019.