DEEP SEQUENTIAL MODELS FOR TWITTER THREADS SENTIMENT ANALYSIS

CSCI 657 PROJECT

Hongbo (Sean) Du
hdu05@nyit.edu
1338320

December 5, 2023

Contents

1 Introduction **1**

2 Preprocessing **1**

2.1 Retrieve and split train set, validation set, and test set 1

2.2 Tidying text and tokenization 1

2.3 Word embedding 1

2.4 Lexicon embedding 2

2.5 Sequence length selection 2

3 Methodology **3**

3.1 Bi-LSTM 3

3.2 Bi-RNN 3

3.3 Training 3

4 Evaluation **4**

4.1 Bi-LSTM 4

4.2 Bi-RNN 5

5 Conclusion **6**

1 Introduction

This project studies the sentimental analysis of two popular models for sequential data, Recurrent Neural Network (RNN) introduced by Elman (1990) and Long Time Short Memory (LSTM) proposed by Hochreiter and Schmidhuber (1997) on the dataset of Twitter threads created by Cacharron (2022). The dataset contains a training dataset of size 150k threads of Twitter posts and comments, and a test dataset of size 60k threads. Both datasets are formatted as a column of true sentiments encoded as 1 for positive and 0 for negative. and a column of the text threads. The files are stored in `.txt` format. For example, a thread is stored as : `0 Starting back at work today Looks like it'll be raining for the next couple of days .`

2 Preprocessing

2.1 Retrieve and split train set, validation set, and test set

Since the dataset is mainly text-based, it is necessary to preprocess the dataset by using natural language processing methods. First, we split the training set and validation set from the original training set. Since the original training set is too large, with a total of 211,983 texts, we only retrieved 8,000 threads for the training set and 2,000 threads for the validation set. Then we retrieve 2,000 threads from the original test set for testing the models. We also merged the original training set and test set for building the complete corpus. Meanwhile, we split the integer labels and the text for both datasets.

2.2 Tidying text and tokenization

we utilize regular expressions in Python to eliminate all punctuation and symbols from the dataset. Punctuation is often regarded as noise in textual data and our goal is to minimize the textual noise. Next, we convert all the letters into lowercase to reduce the size of the word bank. Lastly, we tokenize each word in each text thread, that is, we transform a sequence of text into a list of individual words.

2.3 Word embedding

We use the complete corpus that we previously built to train the word embedding model. We use the Word2Vec skip-gram model, developed by Goldberg and Levy (2014) from Google, to embed each word. The key idea of the skip-gram model is to predict the surrounding words given the current word. For the unseen words, we use a zero vector to embed them.

The resulted word embedding is of size `[length(seq), 399]`. The word embedding result is good. for example, the word 'omg' has the most similar words: {omfg, ahhh, omggg, . . . }.

2.4 Lexicon embedding

We used a lexicon embedding model (Hu and Liu (2004)) to enhance the word embedding performance. We encode each word as 1 if it is negative; 2 as positive; and 0 if the word is not in the lexicon dictionary. Then we perform the naive concatenation (Koufakou and Scott (2020)) to concatenate the Word2Vec embedding with lexicon embedding. As a result, our word embedding is of size `[length(seq), 400]`.

2.5 Sequence length selection

Since the sequences have various lengths, we plotted a sequence length graph to select the sequence length. According to the figure, we can see that a length of 28 can cover most of the sequences. If the length is less than 28, we pad 0 vector to fulfill the length up to 28, otherwise, we truncate the sequence. At this stage, our training set is of size `[8000, 28, 400]`, and 2,000 for both the validation set and test set.

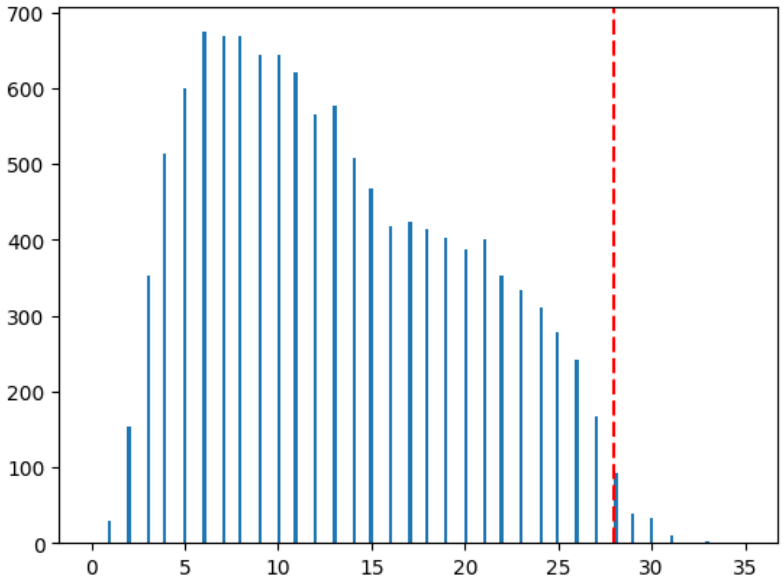

Figure 1: Sequence lengths plot

3 Methodology

3.1 Bi-LSTM

In this study, we constructed a Bidirectional Long Short Time Memory model (Bi-LSTM) (Schuster and K. (1997)). The Bi-LSTM model is structured by two LSTM (Hochreiter and Schmidhuber (1997)) layers, one is forward latent states and the other one is backward latent states. Each layer takes on word embedding tensor, such that $V = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{28}\}$, and as described above, $\mathbf{v}_i$ for $i \in 1, \ldots, 28$ is of size 400. Each word embedding tensor is fed into the forward LSTM layer, and it is also fed into the backward LSTM layer. Then we use a fully connected layer to produce the final output.

3.2 Bi-RNN

Bidirectional Recurrent Neural Network (BRNN) introducted by Schuster and K. (1997) can overcome the limitations of a regular RNN model. Similar to the above model, the model is structured as a part for positive time direction, (forward states) and another part for negative time direction (backward states). Outputs from the two states are distinctive. The hidden size for BRNN is 32 with a dropout rate of 0.5 for avoiding overfitting. Our model only has 1 hidden layer.

3.3 Training

For the training process, we employed Adam optimizer proposed by Kingma and Ba (2014). The Adam optimizer has the features of a self-adaptive learning rate and momentum for overcoming the saddle point problem in optimization. We train our models 15 epochs for Bi-LSTM and 20 epochs for BRNN with a learning rate of $1e-3$, and our minibatch size is 256.

4 Evaluation

4.1 Bi-LSTM

The Bi-LSTM model yields a training loss of 0.00159 and validation loss of 0.00214 at the 15th epoch with a training accuracy of 0.95312 and validation accuracy of 0.77404. Below are the figures for the training accuracy and validation accuracy, and for the training loss.

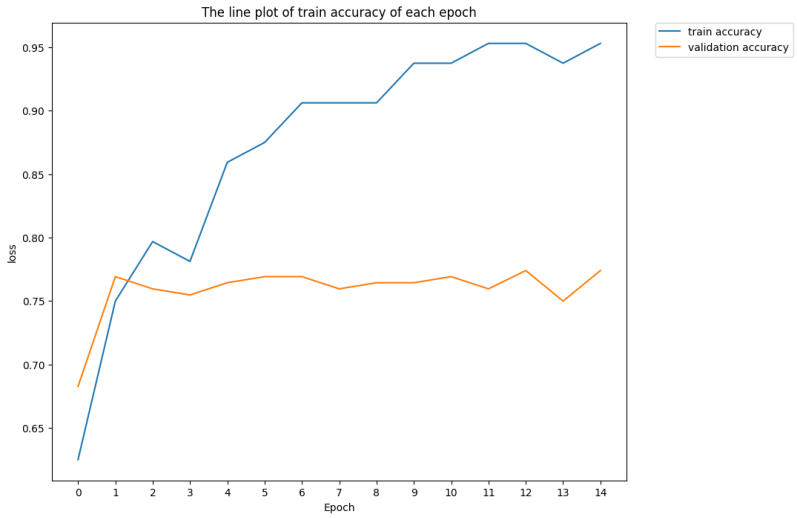

Figure 2: Bi-LSTM accuracy's plot

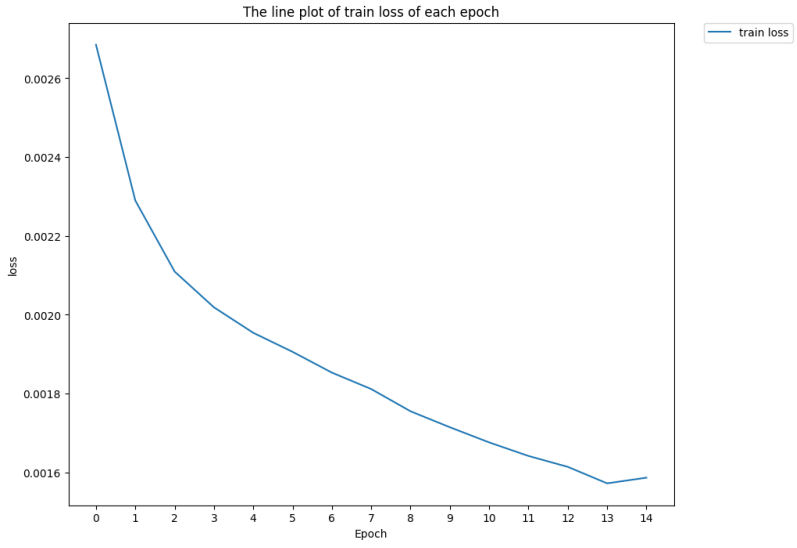

Figure 3: Bi-LSTM training loss plot

The test set also shows a decent result. We can see that the overall accuracy reaches 0.75, 0.7760 for the class 0 precision, and 0.7277 for the class 1 precision. The recall scores are

0.7099 and 0.7909 for the two classes respectively. As a result, the f1-scores are 0.7415 and 0.7580.

Classification report of the Bi-LSTM model on the test set

	precision	recall	f1-score	support
0	0.7760	0.7099	0.7415	1010
1	0.7277	0.7909	0.7580	990
accuracy			0.7500	2000
macro avg	0.7518	0.7504	0.7497	2000
weighted avg	0.7521	0.7500	0.7496	2000

Figure 4: Classification report of the Bi-LSTM model on the test set

4.2 Bi-RNN

The Bi-RNN model yields a lower training loss of 0.00149 and a slightly higher validation loss of 0.00253 at the 20th epoch with a training accuracy of 0.95312 and a validation accuracy of 0.75000. Below are the figures for the training accuracy and validation accuracy, and for the training loss.

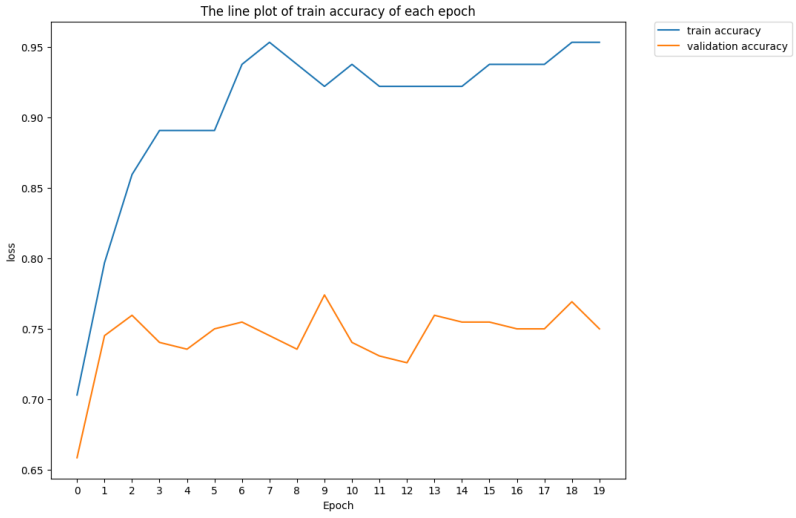

Figure 5: Bi-RNN accuracy's plot

The test set also shows a decent result. We can see that the overall accuracy reaches 0.7200, 0.7023 for the class 0 precision, and 0.7421 for the class 1 precision. The recall scores are 0.7733 and 0.6657 for the two classes respectively. As a result, the f1-scores are 0.7361 and 0.7018.

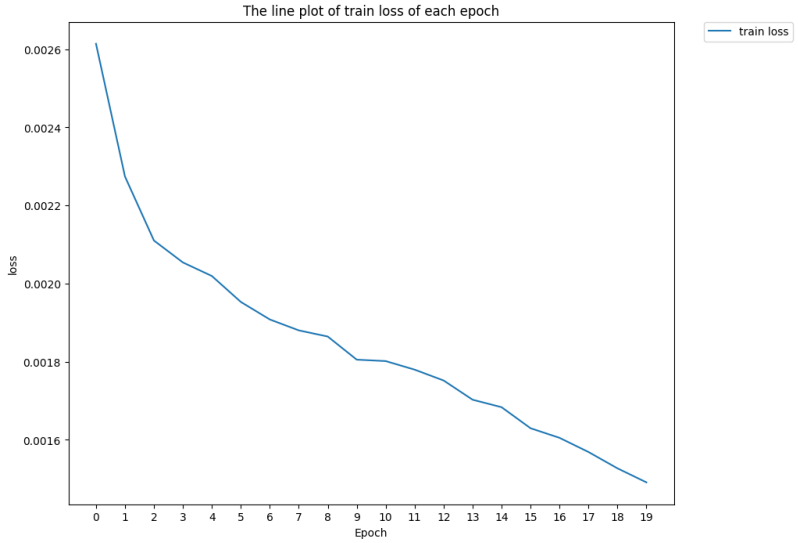

Figure 6: Bi-RNN training loss plot

Classification report of the Bi-RNN model on the test set

	precision	recall	f1-score	support
0	0.7023	0.7733	0.7361	1010
1	0.7421	0.6657	0.7018	990
accuracy			0.7200	2000
macro avg	0.7222	0.7195	0.7190	2000
weighted avg	0.7220	0.7200	0.7191	2000

Figure 7: Classification report of the Bi-RNN model on the test set

5 Conclusion

This project studies the two popular sequential models on the language dataset. We conclude that Bi-LSTM has a slightly better performance than the Bi-RNN model. For future work, we could use the complete dataset to train the models. We could also enlarge the corpus for the word embedding model, and try different word embedding methods, such as Fasttext (Bojanowski et al. (2016)) and GloVe (Pennington et al. (2014)).

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Miguel Carlos Blanco Cacharron. 2022. Sentimentanalysisbert. *Github*.

J. L. Elman. 1990. Finding structure in time. *Cognitive Science*, 14:179–211.

Yoav Goldberg and Omer Levy. 2014. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. Cite arxiv:1402.3722.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Minqing Hu and Bing Liu. 2004. Natural language toolkit: Opinion lexicon corpus reader.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Anna Koufakou and Jason Scott. 2020. Lexicon-enhancement of embedding-based approaches towards the detection of abusive language.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543.

Mike Schuster and Paliwal Kuldip K. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45:2673–2681.