INM427 Coursework. Date of submission 8th May 2022.

Twitter Sentiment Analysis Using Bidirectional Long Short-Term Memory (BiLSTM) and Convolutional Neural Networks (CNN) – A Critical Evaluation

Elnara Mammadova
City University of London, Department of Computer Science
MSc Data Science 2022

ABSTRACT Sentiment analysis is a vastly growing natural language processing (NLP) technique and is widely utilized by numerous sectors in order to assess the sentiment value of their brands, products, or services. This paper provides a critical evaluation of two neural computing models for sentiment analysis on a large-scale twitter dataset. In this paper, we provide an overview of the two deep learning approaches for sentiment analysis tasks, lay out the comparative challenges and provide some critical evaluation. The models considered in this paper are Bidirectional Long Short-Term Memory (BiLSTM) and Convolutional Neural Networks (CNN), supplemented by pretrained GloVe word embeddings and hyperparameter tuning to offer further gains in performance. The experimental results show that the proposed sentiment analysis with BiLSTM method is more effective with higher precision, recall, and F1-score.

INDEX TERMS Sentiment Analysis, Neural Networks, Bidirectional Long Short-Term Memory, Convolutional Neural Networks, Natural Language Processing

I. INTRODUCTION

With the rapid technological development over the last decade, Internet has become the universal source of information for millions of people. Nowadays, more and more people turn to social media where they can freely express their opinions without censorship or restraint. For the first time in history, we have access to a large amount of opinionated data recorded in digital form, ready to be analyzed for sentiment and emotional tendencies through user generated text. Since 2000, sentiment analysis (also known as opinion mining) has grown to be one of the most active research areas in natural language processing (NLP) which is the computational study of people's sentiments, emotions and attitudes towards products or events [1].

Over the years, a lot of studies exploited simple machine learning methodologies to solve sentiment analysis tasks from different perspectives. However, in recent years, deep learning approaches emerged as a powerful substitute which is capable at discovering intricate semantic representations of texts without much feature engineering [1]. Even though in late 1900s research community discounted neural network capabilities due to it being computationally expensive and complex for that time, in the past 10 years deep learning has resurfaced as a dominant machine learning technique in many application domains, more recently in NLP [3][4]. Recent studies have produced state-of-the-art results through deep neural networks which are capable of learning from multiple layers of features from user generated texts.

This study will employ a large-scale open-source twitter dataset to computationally identify sentiment polarity (positive or negative) of the comment corpus towards a non-particular topic.

In this paper, a sentiment analysis method of twitter comments based on BiLSTM, and CNN deep learning models are proposed. Firstly, the research backgrounds of both neural network models are expounded. Later, the details of the proposed word representation method are outlined. Ensuing, the experiments are carried out using both neural network models and the experimental results

are analyzed and discussed. Finally, the proposed methods are summarized and compared, and critical evaluations are being provided.

II. BIDIRECTIONAL LONG SHORT-TERM MEMORY (BiLSTM) NEURAL NETWORKS

LSTM network is a type of RNN (Recurrent Neural Network) capable of learning long-term dependencies where it can capture contextual information by maintaining a state of all previous inputs [5]. For instance, as one reads this paper, one understands each word based on their understanding of the previous words, as the learning process is sequential. Traditional neural networks can't do this, while LSTMs have loops allowing information to persist, and can learn how and when to forget.

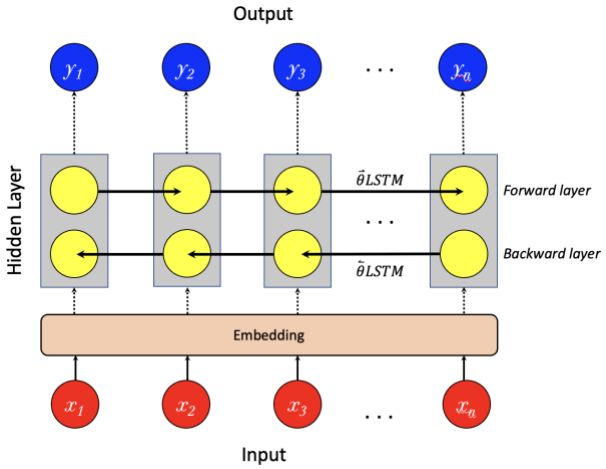

FIGURE 1. **Graphical representation of BiLSTM networks.**

A Bidirectional LSTMs on the other hand is a sequence processing model composed of forward and backward LSTMs (Figure 1). We can generalize BiLSTM as an improved version of LSTM where it can encode information from front to back and back to front and is generally believed to be faster than one-directional approach as it can see future words. BiLSTM can effectively increase the amount of context available to the algorithm by using available input information in the past and future of specific time frame [6].

BiLSTM is superior to vanilla RNN as it doesn't have the same limitation where both input and output has the same size. One of the disadvantages of BiLSTM however is that since it has double LSTM cells it can be costly to train. Additionally, some research studies have shown that it is not a good fit for speech recognition.

III. CONVOLUTIONAL NEURAL NETWORKS (CNN)

CNNs are designed to process structured arrays of data which are generally used for image classification; however, it has subsequently shown to be effective in NLP for text classification problems. One of the earliest applications of CNN in NLP was introduces by Kim Y. (2014) where he demonstrated how CNNs can be used as feature extractors that encodes semantic features of a sentence before feeding them to a classifier [7]. Many since then, have achieved excellent results on sentiment classification problems using only simple one layer CNN trained on top of the pretrained word vectors and very little hyperparameter tuning.

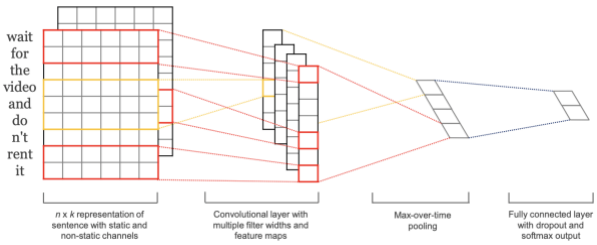

FIGURE 2. **Graphical representation of CNN networks [7].**

CNN uses multiple convolutional filters which is applied to a kernel of n words to produce multiple features. By varying the size of the kernels CNN can automatically detect patterns of multiple sizes (i.e., 2, 3 or 5 adjacent words) without the need of representing the whole vocabulary. After applying each possible window of words, a feature map is generated. The most important features are later extracted from feature maps using max-pooling operation taking the max. value as the feature corresponding to each filter (Figure 2). Use of max-pooling layer after a convolutional layer reduces the output's dimensionality by preserving only the most significant features in the entire sentence.

CNNs offer significant speed advantage in training times over the traditional neural networks, however more recently they have been swept out by visual transformers that offer higher speeds for training. One other disadvantage for CNNs is that unlike LSTMs they don't have a sense of memory state, therefore can be rather limited for sequential modelling. Additionally, CNNs' have difficulty in modelling long word relationships, however it has shown great accuracies in literature where the input dataset contained short sentences.

III. HYPOTHESIS STATEMENT

Due to the ability of BiLSTMs to capture long-term dependencies we believe this model will outperform CNN's predictions. Additionally, since CNNs lack the recurrence of RNNs, they will be unable to keep track of the location of words within a sentence, possibly resulting in knowledge loss. Furthermore, as CNNs don't offer memory states it can offer additional limitations in our sentiment analysis. But since we will be using short sentences as inputs, we expect CNN to have similar accuracy scores to BiLSTM.

III. DATASET

The dataset used in this study is an open-source twitter dataset obtained using Twitter Search API by Stanford University [8]. The sentiments (positive and negative) were classified using distant supervised learning by use of emoticons [9]. Raw data file has 1,600,000 tweets, containing no emojis. The dataset had no prior preprocessing hence, we had to perform extensive text preprocessing to make it suitable for word embeddings and training using neural networks and to achieve higher accuracies. Performed text pre-processing steps are as follows:

1. Remove usernames and html tags
2. Case folding
3. Remove html escape tags
4. Replacing text emoticons with their text representations
5. Replace contractions
6. Pre-process hashtags
7. Pre-process long words
8. Fix for duplicate consecutive letters >2
9. Expanding chat word abbreviations
10. Remove punctuations and empty tweets
11. Tokenization
12. Replace user exclaimed texts with their meaning
13. Remove stop words
14. Lemmatization

After all the text pre-processing visual exploratory data analysis was performed. As seen in the Figure 3, when looking at most common words table, "laugh" occurred most often (almost present in ½ of all the tweets). We will remove this token from our corpus since it may act as noise and negatively impacts model performance. Additionally, most of the "laugh" word was a result of emoticon and chat word expansions during text pre-processing (i.e. ☺, LOL, which are commonly used texts in tweets).

	Most Common Words	count
0	laugh	479888
1	alright	192590
2	good	117185
3	going	90184
4	ready	82534
5	like	79757
6	love	72976
7	work	68745
8	want	65766
9	cannot	64194

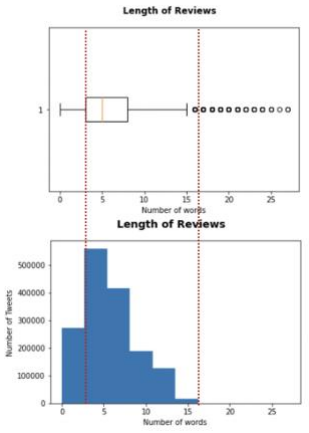

FIGURE 3. **EDA after text pre-processing**

Additionally, looking at the histogram of length of reviews in Figure 3, we can observe that most tweet lengths range between 0-15. Since 0-3 words does not provide much context for the models, we also removed tweets with length less than 3 words, and length greater than 15 words. Figure 4 is a world cloud representation of the final pre-processed data that will be fed into neural network models.

FIGURE 4. **Word Cloud of tweets after text pre-processing.**

III. DESCRIPTION OF CHOICE OF TRAINING AND EVALUATION METHODOLGY

After initial text preprocessing further data preparations were implemented. 50,000 rows were randomly sampled from pre-processed data (to reduce computational cost) and was split into training, validation, and test set in the ratio of 72/18/10 percent respectively. As seen in the Table 1, all three sets are representative sample of the data where the target is evenly distributed (no biased split)

TABLE I
TRAIN-VALIDATION-TEST SPLIT

Dataset	Length	Label distribution	
		Negative	Positive
Training	36000 (72 %)	18288	17712
Validation	9000 (18 %)	4649	4351
Test	5000 (10 %)	2561	2439

As suggested by the PyTorch documentation, dataset class was implemented on all three sets, followed by transformation of list of samples into batches using `DataLoader()` and `collate_fn()` functions. Additionally, since the device processes the training data in batches, the sequences in our batch should have the same length. Hence, we pad our sequence from either side to match the length of the longest sequence in our corpus.

Embeddings learned from datasets with large volume of rare words would fail at arriving at the right representation of the words. To achieve right representations, the dataset must contain a rich vocabulary. Hence word-to-index vocabulary was generated after excluding rare words (all the words that occurred less than 3 times), bringing the total number of words in our vocabulary to 6629.

Pretrained GloVe embeddings were utilized in both neural network models in order to better capture the semantic and syntactic meanings of the words since the embeddings were initially trained on millions of words. Unlike Google's Word2Vec, Stanford's GloVe pre-trained word vectors contain co-occurrence matrices that tells us how often particular word pair occurs together [14]. Even though there was no sparsity of training data, the experimental results on pretrained word embeddings showed much better results compared to model without pretrained vectors. We have also managed to reduce the large number of trainable parameters, by keeping the word vectors static and learn only the other parameters of the model. For instance, while the number of trainable parameters for BiLSTM without pretrained word embeddings returned 1,325,800, with Stanford's GloVe embeddings we reduce that number to 51,200. Number of trainable parameters increase when learning embeddings from scratch, subsequently leading to slower training process. Additionally, we will see when comparing the models with and without pretrained embeddings, that there is a clear performance boost in loss, accuracy and f1 scores.

The GloVe word embedding used in the study is from glove.twitter.27B.200d.txt (dimensionality of word vectors is 200) file which contains the word and embedding vector pair.

IV CHOICE OF PARAMETERS

A. LOSS FUNCTION

Neural networks use optimization process which requires loss function in order to calculate the model error. Binary Cross Entropy (BCE) Loss which is a commonly used cost function for binary classification problem won't work in our sentiment classification problem since it expects the model output and the target to have the same shape. Based on our provided shapes, our target has the shape of batch size and contains class indices for each sample, which is why we are dealing with a multi-class classification use case instead, where Cross Entropy Loss was employed.

A. OPTIMIZER

The choice of optimizer for this problem was AdamW (Adam with decoupled weight decay), for several reasons: (1) AdamW tends to oscillate less than conventional Stochastic Gradient Descent (SGD), (2) AdamW gives better generalizing models in comparison to Adam where L2 regularization is not equivalent to weight decay, (3) AdamW enables you to use larger learning rates per iteration, (4) AdamW converges much faster, and can overcome getting stuck in saddle points.

A. DROPOUT PROBABILITY

Dropout is a regularization method where the key idea is to probabilistically drop out nodes along with their connections from the neural network during training. This improves the generalization of the networks while also reducing overfitting by preventing units from co-adapting too much [10]. As in any other regularization methods dropout is more effective where there is limited amount of training data. For very large datasets, however, regularization confers little reduction in generalization error as the models are less likely to overfit and may see less benefit from using the dropout rate hyperparameter [11]. During the initial fine-tuning with drop-out rates, it was confirmed that dropout rates showed very little change in validation results. Hence, dropout was set to a fixed value of 0.5 and no further hyperparameter tuning was performed on this parameter during grid-search stage.

A. WEIGHT DECAY

Weight decay is another regularization method where it applies linear penalty whereas dropout can cause penalty to grow exponentially [12]. Both generalization techniques are not mutually exclusive, therefore combining dropout and weight decay has become the industry standard when it comes to deep neural network models. The optimal weight decay depends on the total number of batch passes/weight updates [13]. The value for weight decay was calculated using the below equation:

$$\lambda = \lambda_{norm}\sqrt{\frac{b}{BT}}$$

where,

b – batch size,
B – Total number of training samples
T – Total number of epochs.

In the paper by Loshchilov et al (2017) authors found optimal value ranges for λ_{norm} to be between 0.025-0.05. Taking this range for λ_{norm} as a baseline we performed a grid search on both models using hyperparameters values [0.025, 0.04, 0.05].

A. LEARNING RATE SCHEDULER

As discussed in the original paper by Loshchilov et al (2017), the fact that we are using adaptive gradient algorithm

(AdamW) which adapts the learning rate for each parameter, using a global learning rate multiplier can substantially improve the model performance [13]. In this study we decay learning rate of each parameter group by gamma of 0.1 every epoch.

IV EXPERIMENTAL RESULTS AND ANALYSIS

A. BiLSTM RESULTS

BiLSTM layer was built using embedding layer which acts as a lookup table, with words as keys, and vectors as values, followed by a dense linear layer. Dropout layer was applied to help regularize and prevent possible overfitting. `leaky_relu()` activation function (similar to `relu()`, only values are between 0.001 and any other number) was applied to the hidden layer after left and right context were concatenated. Hyper-parameter tuning was performed on three sets of hyperparameters (learning rate, lambda norm and hidden size with initial values of 0.01, 0.025 and 64 respectively).

```
BiLSTM(
  (embedding): Embedding(6629, 200)
  (lstm): LSTM(200, 64, num_layers=2, batch_first=True, dropout=0.5, bidirectional=True)
  (Linear): Linear(in_features=128, out_features=64, bias=True)
  (Dropout): Dropout(p=0.5, inplace=False)
  (output): Linear(in_features=64, out_features=2, bias=True)
)
```

Random

Modules	Parameters
embedding.weight	1325800
lstm.weight_ih_l0	51200
lstm.weight_hh_l0	16384
lstm.bias_ih_l0	256
lstm.bias_hh_l0	256
lstm.weight_ih_l0_reverse	51200
lstm.weight_hh_l0_reverse	16384
lstm.bias_ih_l0_reverse	256
lstm.bias_hh_l0_reverse	256
lstm.weight_ih_l1	32768
lstm.weight_hh_l1	16384
lstm.bias_ih_l1	256
lstm.bias_hh_l1	256
lstm.weight_ih_l1_reverse	32768
lstm.weight_hh_l1_reverse	16384
lstm.bias_ih_l1_reverse	256
lstm.bias_hh_l1_reverse	256
Linear.weight	8192
Linear.bias	64
output.weight	128
output.bias	2

Total trainable parameters: 1569706

Static

Modules	Parameters
lstm.weight_ih_l0	51200
lstm.weight_hh_l0	16384
lstm.bias_ih_l0	256
lstm.bias_hh_l0	256
lstm.weight_ih_l0_reverse	51200
lstm.weight_hh_l0_reverse	16384
lstm.bias_ih_l0_reverse	256
lstm.bias_hh_l0_reverse	256
lstm.weight_ih_l1	32768
lstm.weight_hh_l1	16384
lstm.bias_ih_l1	256
lstm.bias_hh_l1	256
lstm.weight_ih_l1_reverse	32768
lstm.weight_hh_l1_reverse	16384
lstm.bias_ih_l1_reverse	256
lstm.bias_hh_l1_reverse	256
Linear.weight	8192
Linear.bias	64
output.weight	128
output.bias	2

Total trainable parameters: 243906

Non-static

Modules	Parameters
embedding.weight	1326000
lstm.weight_ih_l0	51200
lstm.weight_hh_l0	16384
lstm.bias_ih_l0	256
lstm.bias_hh_l0	256
lstm.weight_ih_l0_reverse	51200
lstm.weight_hh_l0_reverse	16384
lstm.bias_ih_l0_reverse	256
lstm.bias_hh_l0_reverse	256
lstm.weight_ih_l1	32768
lstm.weight_hh_l1	16384
lstm.bias_ih_l1	256
lstm.bias_hh_l1	256
lstm.weight_ih_l1_reverse	32768
lstm.weight_hh_l1_reverse	16384
lstm.bias_ih_l1_reverse	256
lstm.bias_hh_l1_reverse	256
Linear.weight	8192
Linear.bias	64
output.weight	128
output.bias	2

Total trainable parameters: 1569906

FIGURE 5. BiLSTM model structure with total trainable parameters per each user case.

BiLSTM model was tested on three use cases: (1) random – where the embedding layer was randomly initialized and then updated during training, (2) static – where the GloVe pre-trained word embeddings were kept static (3) non-static – where additional backpropagation was performed on GloVe pre-trained word embeddings. The training and validation results were compared for comparison and model selection before any optimization. As seen in Figure 5, static model had the least number of trainable parameters compared to two other user cases.

Figure 6 shows the model performance comparison for training and validation for all three models before any hyperparameter optimization. As seen in the figure, static and non-static models show very similar performance in terms of loss, accuracy and f1-score, compared to random case, where we observe even though both validation and training decrease to a point of stability there is a huge gap between the curves on all three metrics. This may indicate either our network structure being too small for the complexity of the dataset, or we don't have sufficient amount of training data for our model to learn on.

TABLE 2
BILSTM MODEL PERFORMANCE COMPARISON

BiLSTM Model	CPU time	Best Epoch	
		Training	Validation
Random	43s	Loss: 0.44 Acc: 79.7 F1- score:79.4	Loss: 0.6035 Acc: 0.72.1 F1-score: 71.7
Static	40s	Loss: 0.4985 Acc: 75.7 F1- score:75.3	Loss: 0.5225 Acc: 73.6 F1- score:73.2
Non-static	41s	Loss: 0.496 Acc: 75.7 F1- score: 75.3	Loss: 0.5249 Acc: 73.7 F1- score:73.3

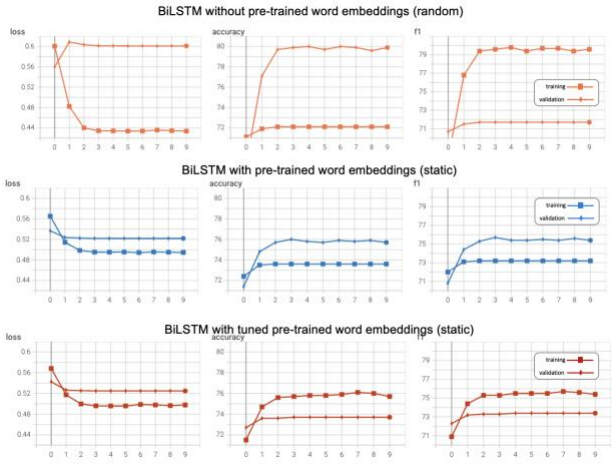

FIGURE 6. BiLSTM model training performance comparison for random, static and non-static user cases.

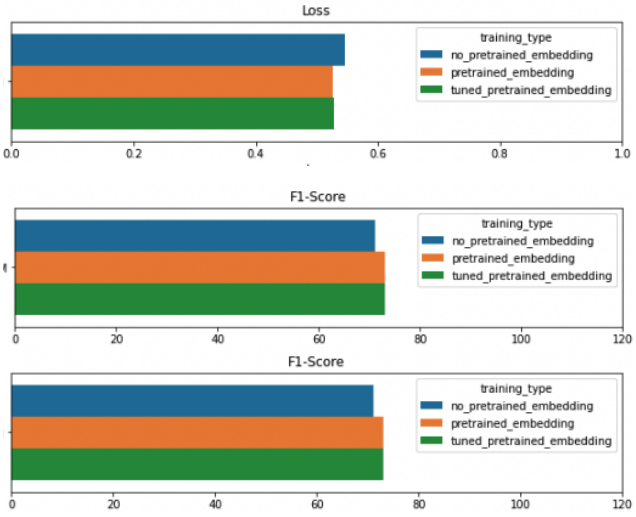

FIGURE 7. BiLSTM model test performance comparison for random, static and non-static user cases.

Table 2 shows the general overview of the training performance comparison of each model where all three models shows similar CPU times. Figure 7 shows the histogram of loss, accuracy and F1-score results for the test

set, where we see little to no difference between static and non-static model, while random model gives poorest metrics out of all three. Based on all the information provided, we proceeded with fine-tuning the final model using static case, and hyperparameters listed below (best parameters marked red):

learning_rate (α) = [0.01, 0.001, 0.0001]
hidden_size = [32, 64, 128]
lambda_norm (λ_{norm}) = [0.025, 0.04, 0.05]

A. CNN RESULTS

Conv2d layers were built, using PyTorch's `ModuleList()` function where all layers were created at once. No hidden layers were applied between convolutional layers and output layer. All convolutional layers were concatenated and fed into the output layer using `nn.Linear()` function. During forward pass, the input text was fed into embedding layer where the shape of our input changed to ["batch size" = 32, "sequence length" =16, "embedding dimension" = 200]. Since convolutional layers require their inputs to be represented in 4 dimensions ("batch", "channel", "height", "width"), embedding layer shape was transformed using `unsqueeze()` function, where one additional dimension was added ("channel" = 1). "Height" and "width" are equivalent to embedding layers "sequence length" and "embedding dimensions" respectively. After applying convolutional layers we used `F.relu()` function and removed the channel dimension from the input (since it is not wanted by the Max Pool). During Max Pooling stage the second dimension ("width") is pooled (empty), hence it is further removed from the layer dimension using `squeeze()` function. Lastly, a dropout layer is applied to the output after concatenating all the convolutional layers. Additionally a custom function was used to pad any input sequence that was less than the filter size before feeding the text to the model. Hyper-parameter tuning was performed on four sets of hyperparameters (learning rate, lambda norm and number of filters and filter sizes with initial values of 0.01, 0.025, 32 and [2, 3, 5] respectively).

```
CNN(
  (Embedding): Embedding(6629, 200)
  (convs): ModuleList(
    (0): Conv2d(1, 32, kernel_size=(2, 200), stride=(1, 1))
    (1): Conv2d(1, 32, kernel_size=(3, 200), stride=(1, 1))
    (2): Conv2d(1, 32, kernel_size=(5, 200), stride=(1, 1))
  )
  (output): Linear(in_features=96, out_features=2, bias=True)
  (Dropout): Dropout(p=0.5, inplace=False)
)
```

Random Static Non-static

Modules	Parameters
Embedding.weight	1325800
convs.0.weight	12800
convs.0.bias	32
convs.1.weight	19200
convs.1.bias	32
convs.2.weight	32000
convs.2.bias	32
output.weight	192
output.bias	2

Total trainable parameters: 1390090

Similar to BiLSTM model CNN was tested on three use model cases (random, static, non-static). The training and validation results were compared for comparison and model selection. The trainable parameters for all three cases were 1,390,090 (unlike BiLSTM, CNN's trainable parameters does not reduce with static model). As seen in Figure 8, results from the baseline model without pre-trained word embeddings the plot for both training and validation loss decreases to a point of stability, however with a big gap between them. This may be an indication that training dataset has too few examples and does not provide sufficient information to learn the problem. For the non-static user case where the pre-trained word embedding layer is fine-tuned during training, we can see somewhat of a flat lines for both validation and training are too flat, which may mean the model does not have the suitable capacity for the complexity of the dataset to learn from. CNN model for the static case shows the best performance, where both training and validation decrease to a point of stability and with very little gap with them. However, continued training on this model may likely lead to an overfit.

Table 3 shows the general overview of the training performance comparison of each model static model outperforms others in CPU times.

TABLE 3
CNN MODEL PERFORMANCE COMPARISON

BiLSTM Model	CPU time	Best Epoch	
		Training	Validation
Random	4min 38s	Loss: 0.5138 Acc: 75.3 F1- score: 74.6	Loss: 0.6056 Acc: 69.8 F1-score: 69.0
Static	1min 28s	Loss: 0.5618 Acc: 71.7 F1- score:71.1	Loss: 0.5608 Acc: 71.6 F1- score:70.9
Non-static	4min 35s	Loss: 0.5577 Acc: 71.5 F1- score: 70.8	Loss: 0.5607 Acc: 71.6 F1- score:70.9

*CNN was trained on CPU due to GPU limitations on Google Colab

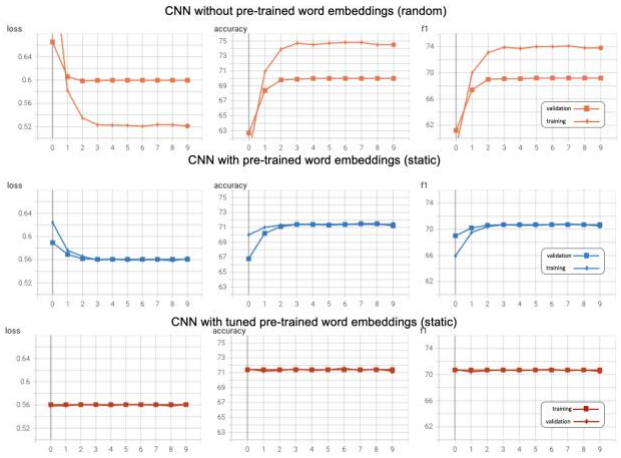

Figure 10 shows the histogram of loss, accuracy and F1-score results for the test set, where we see little to no difference between static and non-static model, while random model gives poorest metrics out of all three. Based on all the information provided, we proceeded with fine-tuning the final model using static case, and hyperparameters listed below (best parameters marked red):

learning_rate (α) = [0.01, 0.001, 0.0001]
n_filters = [32, 64]

filter_sizes = [[2, 3, 5, 7], [2, 3, 5]]
lambda_norm (λ_{norm}) = [0.025, 0.04, 0.05]

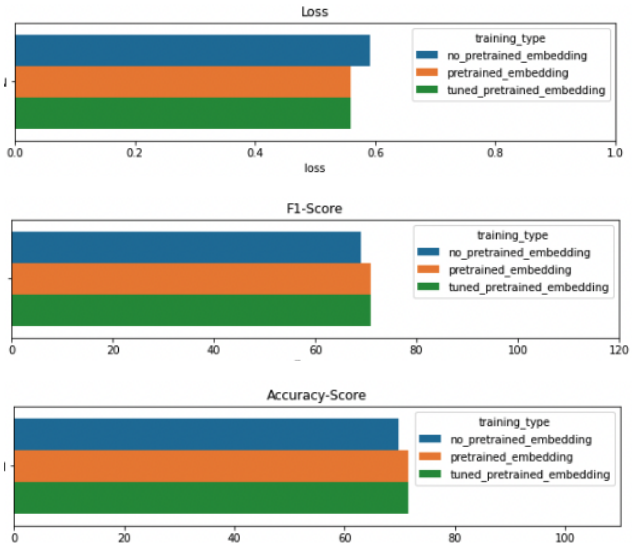

FIGURE 10. CNN model test performance comparison for random, static and non-static user cases.

IV CRITICAL EVALUATION

The hyperparameter tuning was performed using custom grid-search with early stopping modification (stop training after 5 epochs if no improvement seen since last best epoch). Proceeding with the tuned hyperparameters, the new models were trained and tested. Figure 11, shows the confusion matrix for test predictions for CNN (blue) and BiLSTM (red). As seen from the confusion matrix, BiLSTM gives better predictions for both sentiments, proving our initial hypothesis.

The predictions for both models could be improved further by increasing the amount of data. Due to the computational limitations this study had to be done on a smaller subset (50K tweets), however the original file contains 1.6 million tweets for training set alone. For CNN it seems like our network architecture is too small. Additionally, even though dropout is a regularization technique that is most effective at preventing overfitting putting it right before the last layer in the CNN is probably not ideal. This is most probably a bad place to apply dropout, since network has no ability to correct the error produced by the dropout layer before the classification takes place. In BiLSTM dropout might have also been avoided since the network is small relative to the dataset. Since the model capacity is already low lowering it further by adding a dropout regularization on top of weight decay introduced by AdamW optimizer might have hurt the performance further.

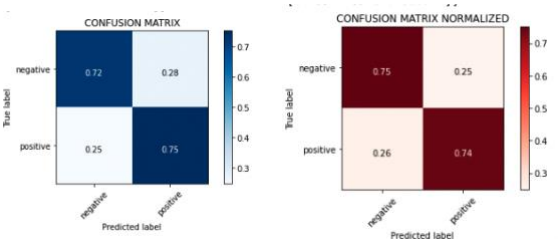

FIGURE 11. Confusion matrix for test predictions for CNN(blue) and BiLSTM (red).

REFERENCES

[1] Tang, D., Qin, B. and Liu, T. (2015) "Deep learning for sentiment analysis: successful approaches and future challenges: Deep learning for sentiment analysis," *Wiley interdisciplinary reviews. Data mining and knowledge discovery*, 5(6), pp. 292–303.

[2] Xu, G. et al. (2019) "Sentiment analysis of comment texts based on BiLSTM," IEEE access: practical innovations, open solutions, 7, pp. 51522–51532.

[3] Collobert, R. *et al.* (2011) "Natural Language Processing (almost) from Scratch,"

[4] Goldberg, Y. (2016) "A primer on neural network models for natural language processing," *The journal of artificial intelligence research*, 57, pp. 345–420

[5] Hochreiter, S., Schmidhuber, J. (1997) "Long Short-term Memory," *Neural computation*. 9, pp. 1735-1780

[6] Schuster, M., Paliwal, K. (1997) "Bidirectional recurrent neural networks," *Signal Processing, IEEE Transactions*, 45, pp. 2673 – 2681

[7] Kim, Y. (2014) "Convolutional Neural Networks for Sentence Classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Stroudsburg, PA, USA: Association for Computational Linguistics.

[8] *For Academics - Sentiment140 - A Twitter Sentiment Analysis Tool* (no date) *Sentiment140.com*. Available at: http://help.sentiment140.com/for-students/

[9] Go, Alec & Bhayani, Richa & Huang, Lei. (2009) "Twitter sentiment classification using distant supervision," *Processing*, 150.

[10] Srivastava, N. *et al.* (2014) "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*. 15. pp. 1929-1958.

[11] Goodfellow, I., Bengio, Y., Courville, A. and Bach, F., 2016. Deep Learning (Adaptive Computation and Machine Learning Series) The MIT Press. *Cambridge, MA, USA*, p.800.

[12] Helmbold, D.P. and Long, P.M., 2017, June. Surprising properties of dropout in deep networks. In *Conference on Learning Theory* (pp. 1123-1146). PMLR.

[13] Loshchilov, I. and Hutter, F., 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

[14] Pennington, J., Socher, R. and Manning, C. (2014) "Glove: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Stroudsburg, PA, USA: Association for Computational Linguistics.

APPENDIX A. GLOSSARY

Accuracy - The fraction of predictions that a classification model got right. In a binary classification accuracy has the following definition:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Number Of Examples}}$$

Binary classification - A type of classification task that outputs one of two mutually exclusive classes.

Class - One of a set of enumerated target values for a label.

Classification Model - A type of machine learning model for distinguishing among two or more discrete classes.

Confusion Matrix - An (N x N) table that summarizes how successful a classification model's predictions were; that is, the correlation between the label and the model's classification. One axis of a confusion matrix is the label that the model predicted, and the other axis is the actual label. N represents the number of classes.

Dataset - A collection of examples

Hyperparameter - The "knobs" that you tweak during successive runs of training a model.

Label - In supervised learning, the "answer" or "result" portion of an example. Each example in a labelled dataset consists of one or more features and a label.

Loss - A measure of how far a model's predictions is from its label. Or, to phrase it more pessimistically, a measure of how bad the model is.

Loss curve - A graph of loss as a function of training iterations.

Machine Learning - A program or system that builds (trains) a predictive model from input data. The system uses the learned model to make useful predictions from new (never-before-seen) data drawn from the same distribution as the one used to train the model. Machine learning also refers to the field of study concerned with these programs or systems.

Model - The representation of what a machine learning system has learned from the training data

Overfitting - Creating a model that matches the training data so closely that the model fails to make correct predictions on new data.

Prediction - A model's output when provided with an input example.

Long Short-Term Memory (LSTM) - A type of cell in a recurrent neural network used to process sequences of data.

Test set - The subset of the dataset that you use to test your model after the model has gone through initial vetting by the validation set.

Training - The process of determining the ideal parameters comprising a model.

Training set - The subset of the dataset used to train a model.

Pytorch - Open source machine learning framework for deep learning.

Tensor - *n*-dimensional array used for numeric computations. In PyTorch the data is passed to the artificial neural networks, and returned by them, as tensors.

Validation - A process used, as part of training to evaluate the quality of a machine learning model using the validation set. Because the validation set is disjoint from the training set, validation helps ensure that the model's performance generalizes beyond the training set

GridSearch - Technique to search over specified parameter values for a model.

Convolutional Neural Network (CNN) - A neural network in which at least one layer is a convolutional layer.

**Definitions taken from Google Developer's Machine Learning Glossary https://developers.google.com/machine-learning/glossary*

Access to google drive with all the codes and outputs: https://drive.google.com/drive/folders/1l5kbVayw81EVSTGXUXhwo2UFe7w_88U7?usp=sharing

APPENDIX B. IMPLEMENTATION NOTES

The working directory of codes were run on Google Drive using Python 3.7.13. All the additional dependencies were pre-installed using pip function. The test notebooks require to be run fully. Note that since glove embedding file was too large it can be uploaded to Turnitin. Therefore, installation of glove embedding file (glove.twitter.27B.200d.txt) is required which will take some time. Click Run All on the test notebooks (to train and test both models). Training takes less than 2 minutes.

Intermediate Results

(1) Initially we ran all the models with the minimal pre-processing to the dataset (general text pre-processing with punctuation, stop word and contraction removal, cleaning of html and username tags, lemmatization). The predictions were low, and the models failed to generalize well. Further pre-processing was carried out, i.e. hashtag, long-words, chat-words and emoticon pre-processing, which increased our vocabulary size as well as provide a cleaner and more uniform distribution of tokens.

(2) At first custom L2 regularization was applied to both networks, which was then switched to AdamW optimiser with weigh-decay, which added additional parameter for tuning, as well as provide better accuracies.

(3) Learning rate scheduler was applied next, as recommended by the literature for networks with adaptive gradient algorithms, which further increased the performance of our models.

(4) We also experimented with BiLSTM with last layer's hidden states concatenated, which didn't provide any additional improvements to the model performance. Therefore, we resorted to using the initial model.

(5) For CNN we experimented with Conv1d and Conv2d. Even though the literature suggested Conv1d layers generally performing better in NLP classification problems, Conv2d proved to be performing better on our dataset.