Sentiment Analysis on Covid19 texts: The impact of the Bidirectional layer on a Long Short Term Memory Recurrent Neural Network

[Author Name(s), First M. Last, Omit Titles and Degrees]
[Institutional Affiliation(s)]

Author Note

[Include any grant/funding information and a complete correspondence address.]

Abstract

The pandemic crisis erupted by covid19 outburst at the end of 2019 created consequent literature in social media. The analysis of such texts, especially when coming from non-scientific organizations like online news providers, may prove useful to reveal the pandemic's impact in the real world. Moreover, monitoring the variability and the evolution of opinions coming from different sources may inform analysts about opinion centers' existence and people's endurance on hard or mild measures on facing virus spreading. On the other hand, Sentiment Analysis is a well-investigated but not exhausted data science topic welcoming further research and improvement. Following recent innovations in this field, this study analyzed the texts' sentiment in the context of Machine Learning (ML) for Text Classification rather than using predefined values assigned on lexical entities. For this purpose, 12,284 articles consisted of several sentences were processed and labeled for negative or positive sentiment. In the context of Sequence Classification using Deep Learning (DL), the study examined two state-of-the-art algorithms: (a) A Long Short Term Memory Recurrent Neural Network (LSTM) and (b) Extending (a) to a Bidirectional Long Short Term Memory Recurrent Neural Network (BLSTM). The study examined the impact of the Bidirectional extension of LSTM in the flow of changing parameter values like the batch size and dropout rate. The evaluation of these two models regarded the calculation of accuracy in a validation dataset. The outcomes made it clear that the classification of articles from online news providers related to covid19 into positive or negative sentiment using RNN can be successful. Moreover, extending LSTM to BLSTM can be a valuable addition to the DL recipe and an accuracy of 90% is feasible.

Keywords: NLP, LSTM, BLSTM, RNN, Deep Learning, DL, Text Classification, Sentiment Analysis, Data Science, COVID19, coronavirus, Neural Network, RNN

Sentiment Analysis on Covid19 texts: The impact of the Bidirectional layer on a Long Short Term Memory Recurrent Neural Network

The rapid and continuous increase in social media allowed people of all categories to share their opinions about a wide range of topics. Moreover, the same people became keen to read and be influenced by online news providers and opinion leaders. A consequence of these was sentiment analysis to become an important tool that may increase the knowledge and understanding of forming "common sense".

The pandemic of coronavirus led mass media to provide constant updates of the crisis describing the impact on the economy and society and the innovations in the medical science related to vaccines and treatments. Such articles, when coming from well-known providers, could work like a receiver-transmitter sentiment tool. An economic analyst could read these texts to foresee an economic impact, but a political analyst could write such articles to guide people's sentiment.

In the context of data science, the problem of classifying texts according to sentiment can be solved by Sentiment Analysis using predefined polarity assigned to each word. Though easy to apply with modern Python tools like Vader of NLTK and TextBlob, this tactic could meet several drawbacks. Interaction among semantics, morphology, and colloquialisms belong to a non-conceivable spectrum when using such tools. Moreover, identical words within different language frameworks produce the same polarity, thus skewing what a natural speaker perceives. (Figure 1 demonstrates a case of skewed sentiment.)

```
1 text = "Good heavens!"
2
3 ('TextBlob:', TextBlob(text).polarity ,
4  'nltk.vader:', vader.polarity_scores(text)['compound'])
```

```
('TextBlob:', 0.875, 'nltk.vader:', 0.7088)
```

Figure 1. A Case of Skewed Sentiment

The second solution for a data scientist is to complete this task by constructing a model following the principle "learning from example". This supervised training process aims to estimate the parameters of a problem using training data. Defining the model's structure in advance is the only way for the scientist to use prior knowledge regarding this problem (Schuster et al., 1997).

Recent evolution in computer processors permits data scientists to use as many texts as they can get and train complicated ML models with millions of parameters in an acceptable time. A difficulty that can be arisen by this kind of treatment is that ML models need labeled data. When treating a binary Text Classification problem, one or more annotators should read the texts and assign a positive (1) or a negative (0) value to each one of them (labeling). One may conduct a Sentiment Analysis in the context of Text Classification. After labeling texts with positive or negative sentiment, a trained model with this labeled data can classify unknown unlabelled texts.

There are many different approaches to treat a Text Classification problem. There also many choices to vectorize text and represent it as numbers since a number is what an ML algorithm understands. A simple way is to use the bag-of-words (BOW), a dictionary representation calculating a word frequency where each word is a feature. A more advanced dictionary approach is the n-gram approach, where each gram consists of n continuous words. More advanced structures to represent text could be word embeddings and word sequences. The data scientist may preprocess texts before vectorizing and clean them from stopwords, misspelled words, and non-semantic entities. Words also maybe lemmatized or stemmed using

well-known and tested automated tools like Wordnet Lemmatizer or Porter Stemmer, both also implemented within NLTK.

Bayesian models could treat simple problems by calculating conditional probabilities, while another approach that earned much attention in previous years was the Support Vector Machines (SVM) algorithm. SVM could achieve better results in several cases (Shah et al., 2016). Nowadays, data scientists use Neural Networks (NN) escalating DL. Data scientists may use various NN to treat a Text Classification problem.

This study demonstrates a DL recipe, including layers of Word Embeddings, Convolution, Dropout, and the extension of LSTM to BLSTM. The purpose is to analyze the impact of extending LSTM to BLSTM e in a flow of increasing dropout rates assigned on different layers and achieving progressively higher classification accuracies.

1. Previous work

Journal of Information and Telecommunication recently published a Nemes and Kiss (2020) paper about social media sentiment analysis based on covid19. From Twitter API, they downloaded recent data (tweets) using the 'covid' keyword. With this data, they trained an RNN using the Keras implementation of Tensorflow and classified tweets into seven ordinal categories (from Strongly Negative to Positive). They compared their results to TextBlob's sentiment analyzer, which resulted in a significant difference. The TextBlob analyzer proved biased in favor of the Neutral sentiment category in contradiction to RNN, which was biased against neutrality. Moreover, Twitter sentiment about Covid19 was mainly negative.

Dai et al. (2019) investigated backdoor attacks to a deep neural network. Specifically, they investigated such attacks on the backdoor of RNN. They implemented a backdoor attack against LSTM-based text classification for Sentiment Analysis by poisoning the dataset of IMDB movie reviews. They explain that when one injects the backdoor, the model will missclassifies any text samples that contain a specific trigger sentence into a target category. Their experiment achieved a 96% success rate, with a 1% poisoning rate proving that LSTM is also vulnerable to backdoor attacks like CNN.

Borna and Ghanbari (2019) analyzed the Hierarchical LSTM network (HAN) for Text Classification. They mined three different datasets and trained CNN, RNN, and HAN for each of these datasets. They concluded that CNN is a decent general method for acceptable validation accuracy, while RNN and HAN did not provide consistent results. Nevertheless, they concluded that HAN outperforms other algorithms when treating problems with vast datasets.

Rao and Spasojevic (2016) used LSTM combined with Word Embeddings and developed Text Classification models for more than 30 languages achieving high accuracy (more than 87%). They built two axes of binary classifications. The first one was to detect Actionability, and the second one was to catch the political preference of tweets. They used extensive datasets with more than 300,000 rows, and they concluded that LSTM outperforms other algorithms when combined with Word Embeddings. They also reached a recipe to maximize accuracy that sets LSTM to 32 units, Word Embeddings to 128, and the batch size to 64 observations, adding a Dropout layer before the activation function. Moreover, they investigated the optimizers and the activation function type concluding that ADAM and the sigmoid function increased accuracy.

Hassan and Mahmood (2017) suggested a neural language model on the base of pre-trained word vectors. They utilized the Bidirectional Recurrent Neural Network (BRNN) to substitute pooling layers in CNN to preserve the local information's details and catch long-term dependencies. They validated their model using two notorious datasets: (a) Stanford Large Movie Review (IMDB) and (b) Stanford Sentiment Treebank (SST). They concluded that CNN could extract higher-level features invariant to local translation, and the RNN preserved order information even with one layer and suggested their combination.

2. Background

2.1 Textual data.

2.1.1 Handling Textual Data.

Text Classification problems demand that a data scientist handles text to create a meaningful structure appropriate for numerical transformation. The most common actions are cleaning, tokenization, stopword removal, lemmatization, and stemming (Shah et al., 2016). The purpose of these steps is (a) remove from the text possible features semantically indifferent (cleaning, stopword removal), (b) extract features (tokens) like words, emoticons, periods (full dots, question marks, commas, etc.), or/and (c) normalizing terms by reducing the morphological index to lemma or stem. Algorithms may use detailed dictionaries to draw the appropriate information (Leopold et al. 2002). After the completion of these steps, the remained entities are the features of the classification problem. The presence and the interaction of these features describe the problem's classes (Han et al., 2001).

2.1.2 Vectorizing Text Features.

Natural Language Processing (NLP) enables the communication between computers and humans by building computational models. Such communication is feasible only after equivalent numeric representations replace textual data to be handled by an ML algorithm (Elghannam, 2019). The well-known BOW illustrates such a vectorizing process creating a sparse representation despite not preserving the sentences' structure (Hu et al., 2012).

Scientists using DL may also use Word Embeddings. This representation uses dense vectors to project each word into a continuous vector space. The word's position in the vector space depends on its surroundings. The embedding is the word's position in the learned vector space (Mikolov et al., 2013). Word embeddings create vectors able to keep semantic similarity according to the linguistical distributional hypothesis (Firth 1957).

2.2 Layers of a DL Recipe.

In this study, a DL recipe is the classifier pipeline (Sequential model) after vectorizing text. In the context of the Keras implementation for DL in Python: "A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor." (https://keras.io/guides/sequential_model/).

2.2.1 Convolution Layer.

A convolution layer as part of a DL recipe is an automated system for feature extraction from a fixed-length segment. The location of the feature is not significant (Géron Aurélien, 2019). There are three hyperparameters to tune: (i) The depth to define the number of neurons to be connected, (ii) The stride to determine the allocation of the depth columns, and (iii) The padding to control the spatial size of the output. A convolution layer on the top of a word vector before a fully connected layer can achieve excellent sentence classification results (Kim, 2014).

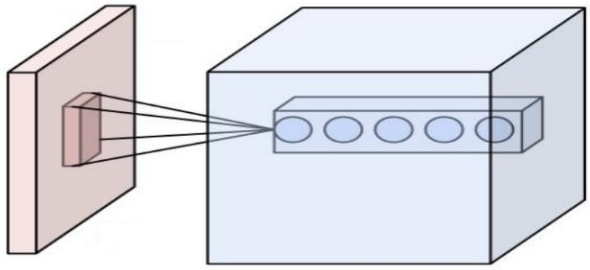


Figure 2. Input volume connected to a convolutional layer
By Aphex34 - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=45659236

2.2.2 Max Pooling Layer.

The Max Pooling layer divides the vector into equal rectangles and then outputs the maximum from every rectangle. The intuition behind this process is that the relativity of the location of a feature is critical. The result of this layer is to control overfitting by reducing the spatial size of the representation. Successive convolution layers activated by a Rectified Linear Unit (ReLU) and followed by a Max Pooling Layer may prove more efficient (Géron Aurélien, 2019).

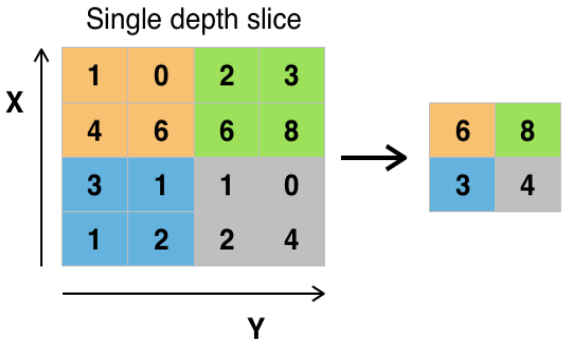


Figure 2. Max_pooling with 2x2 filter and stride = 2
By Aphex34 - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=45673581

2.2.3 LSTM Layer.

LSTM is an improvement in the context of RNN to treat the problem of vanishing and exploding gradients after backpropagation (Hochreiter; Schmidhuber, 1997). LSTM units consist of cells and gates. A cell is the memory of the LSTM unit to record the dependencies of the input sequence. With LSTM units, when error values are back-propagated from the output layer, the error remains in the LSTM unit's cell. There are three gates in every unit to control the information flow: (i) the input gate to control the flow of new values, (ii) the output gate to determine whether the values in memory will be used in the activation, and (iii) the forget gate to remove a value from the cell. The activation can be a sigmoid or a tangent function.

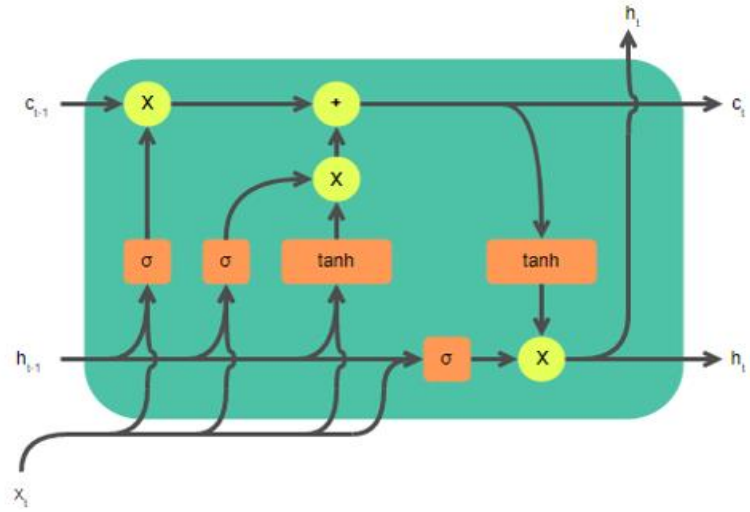


Figure 2. Internal Structure of an LSTM Cell.
Original Version: Guillaume Chevalier, Redrawn as SVG by ketograff - https://commons.wikimedia.org/wiki/File:The_LSTM_cell.png)

In Figure 3, each orange box is an activation function, and each yellow circle is a pointwise operation. Merging two arrows means a linear transformation, while splitting one arrow means a copy operation.

2.2.4 The Bidirectional Extension on LSTM Layer.

Extending an LSTM layer to Bidirectional connects two hidden layers of opposite directions. With this extension, the output layer simultaneously carries information from the past and the future states (Schuster, Paliwal, 1997). The earlier hidden states only observe a few vectors from the lower layer, while the later ones are computed based on most lower-layer vectors (Hassan, Mahmood, 2017).

2.2.5 Dropout Layer.

Dropout in the context of a DL recipe is a regularization process of removing from the NN a percentage of nodes to avoid overfitting and maximizing the model's generalization. The removal of the nodes during dropout is a stochastic process. The reduced network keeps updating their weights, while the removed

nodes preserve their weights during this training stage. The results are (a) accelerated training, (b) feature robustness, and (c) depreciation of nodes' interaction (Srivastava et al., 2014).

2.2.6 Dense Layer.

A Dense layer is a fully connected layer where every input is connected to an output with a weight. The Dense layer performs a linear operation, which may end with a non-linear activation function. Several Dense layers may exist in a DL recipe. However, the last layer in the DL classification tasks calculates the class's probability, and this is a Dense layer with the appropriate non-linear activation. The last layer's usual activation is the sigmoid function (Hassan, Mahmood, 2017), but any function is possible for the previous dense layers.

2.3 An Optimization Algorithm.

There are several optimization algorithms one may select. This selection can make a difference not only for the result to achieve but also for training time. Kingma and Ba (2015) introduced an efficient stochastic optimization that only requires first-order gradients and little memory requirement. Their method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. They named their algorithm ADAM derived from adaptive moment estimation.

Their method combines the advantages of two popular methods: AdaGrad (Duchi et al., 2011), which works well with sparse gradients, and RMSProp (Tieleman & Hinton, 2012), which works well in online and non-stationary settings. In 2017, Ruder (2017) suggested that ADAM might be the overall best choice after comparing a series of famous Stochastic Gradient Descent algorithms. Moreover, Kinga and Ma (2015) experiment on IMDB BoW features suggest that Adam can be more efficient for text classification tasks (Figure 4). Rao and Spacojevic (2016) also supported the ADAM's superiority.

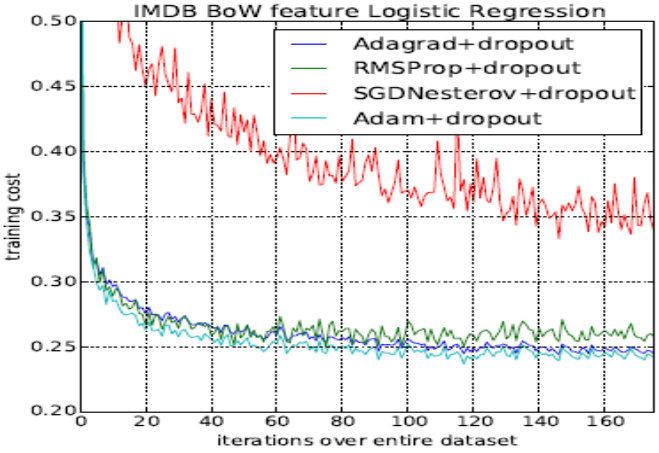


Figure 4. Logistic regression training negative log-likelihood on IMDB movie reviews with 10,000 bag-of-words (BoW) feature vectors (Kingma, Ba, 2015)

3. Methodology.

3.1 Data.

3.1.1 Selecting and Downloading Data.

This study's choice regarding the data was to scrape new, unused public data from famous online news providers using keywords like coronavirus and covid19. A scraper downloaded a total number of about 10.000 articles corresponded to the date range from May until September of 2020. The primary providers of these articles were *10news.com, cnn.com,* and *foxla.com*. The downloaded articles varied significantly in the number of sentences and the word count.

3.1.2 Normalizing Text Sizes.

Many articles too extended in size created a question of truncating or split them. On the other hand, others were too short. The decision was to reconstruct articles to create paragraphs with an approximate length of 10 sentences to utilize as much data as possible. For this procedure, an automated script that used NLTK's sentence tokenizer extracted 41,839 text entities that would be used for sentiment analysis when labeled. The NLTK tokenizer was updated to recognize abbreviations like 'Dr.', 'Mr.', 'Mrs.', 'prof.', 'inc.', and 'i.e.' to avoid some known mistakes.

3.1.3 Labeling Data.

The task of labeling data was handled manually by human annotation. The annotators assigned the value 1 (positive class) for texts with positive sentiment and the value 0 (negative class) for text with negative sentiment. After the end of the labeling process, about 8,500 texts were removed from the dataset. Annotators assigned these texts with contradictory labels because of neutrality or interpretation difficulty. The final dataset contained 33,324 texts, from which 62.51% belonged to the positive class.

3.2 Text Cleaning - Preprocessing.

Despite the formality of the texts, which saved endless hours of dealing with spelling inconsistencies and mistakes, the articles contained various features that added a non-necessary burden on the dimensionality. Moreover, there were names of famous personalities like 'Donald Trump' and other entities that could add bias to the model. It would be easy for a DL model to learn the names and titles connected to a sentiment instead of learning unbiased text features. There were two cleaning axes: (i) clean tokens semantically insignificant, and (ii) clean tokens that may convey bias. Thus, titled names, uppercased words, tokens containing non-alphanumeric characters, numbers, stopwords, and words like 'corona', 'coronavirus', 'covid19' were removed from the text.

This study also introduces an abstraction token to replace 22 negations like not, wouldn't, cannot, can't, mightn't, etc. under the token <negation>. Keeping a negation indicator in the text might help the model to distinguish bigrams and trigrams like "not bad", "not good enough", "cannot do". Then the remained tokens were stemmed using the Porter Stemmer of NLTK.

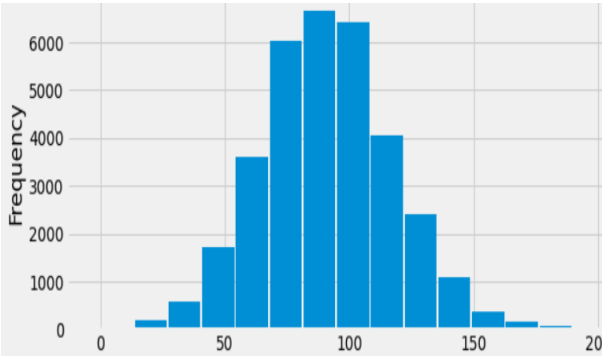


Figure 5. Histogram of tokens number.

The average number of tokens for texts remaining in the dataset was 91, with a maximum of 271 and 75% of the instances to have more than 73 tokens (Table 1). The tokens number follows the Normal Distribution closely except for a minimal number of outliers creating a right tail (Figure 5).

count	33,324.00
mean	91.19
std	26.38
min	-
25%	73.00
50%	91.00
75%	108.00
max	271.00

Table 1. Tokens' Number

The analysis above about tokens number led to the decision to normalize the data further and remove possible outliers. Under this concept, texts containing less than 36 tokens were removed. This token number corresponds to the 0.0150st quantile of the tokens distribution. That way, 1.5% of the data, possibly corresponding to outliers, was excluded.

3.3 Vectorizing Tokens

Data provided now 32,841 sequences of cleaned and stemmed text tokens. The vocabulary of this data reached 22,073 unique tokens. The frequencies of these tokens were calculated and ranked. Also, the minimum acceptable frequency was set intuititevly to eight. The most frequent tokens were the: <said> with 50,237 appearances, <negation> with 28,093, <peopl> with 22,064, <state> 16,358, and <case> with 15,676. The token <negation> corresponds to stopwords preserved in the text under this abstraction token as described in 3.2. Some tokens with minimum frequency were <swoosh>, <investiture>, <unclimb>, <calfir>, <mandarin>.

For constructing the sequences, unique integers replaced every token. Tokens having frequency under the minimum (8) received zero value. The length of every sequence was set to the 0.9850st quantile of the token distribution (152). Thus, after truncating and padding the sequences, the vectorized dataset consisted of 32,841 sequences of 152 units.

3.4 Train Test Split

For this study, 67% of the data would be the training set, and the rest will be the validation set. The split was random, but a seed was selected for reproducibility. The number of sequences to train the model was 22,003, and the number of sequences to test it 10,838.

3.5 Sequential Model.

3.5.1 Strategy.

The sequential model's main structure was due to utilizing the previous work, which has been described in section 1. The building strategy of the sequential model was to add a dropout after every layer. Conducting several experiments with fluctuating numbers of parameters and taking the maximum accuracy after training models differing to the Bidirectional extension of LSTM would create a flow of comparable accuracies.

3.5.2 Baseline Model.

The first layer on the stack would be an Embedding layer with input the vocabulary number (22,073 unique tokens) and output a 64 dimension vector of length 152 (0.9850st quantile of the token distribution). A Dropout layer would follow. The third layer would be a one-dimensional convolution layer with 16 filters and an activation function set to ReLU, followed by a Max Pooling layer with pool size set to two. The fifth layer would be again a Dropout layer, and then the LSTM layer would follow either with Bidirectional extension or not. The LSTM layer also supports internal dropouts, but one more Dropout layer is added that corresponds to the whole system. The system is completed with a Dense layer with a non-linear activation (sigmoid function) to calculate the final output (probabilities), a binary cross-entropy as the loss function, and ADAM's declaration be the DL optimizer. Table 2 summarizes the layers and the parameters of the model.

Model: "sequential_132"

Layer (type)	Output Shape	Param #
embedding_132 (Embedding)	(None, 152, 64)	1412736
dropout_392 (Dropout)	(None, 152, 64)	0
conv1d_131 (Conv1D)	(None, 152, 16)	3088
max_pooling1d_131 (MaxPoolin	(None, 76, 16)	0
dropout_393 (Dropout)	(None, 76, 16)	0
lstm_131 (LSTM)	(None, 32)	6272
dropout_394 (Dropout)	(None, 32)	0
dense_131 (Dense)	(None, 1)	33

Total params: 1,422,129
Trainable params: 1,422,129
Non-trainable params: 0

Table 2. LSTM Baseline Model

3.6 Impact of the Bidirectional Extension.

A way to analyze how BLSTM may impact optimal parameters is to conduct six experiments training LSTM and BLSTM models for five (5) epochs using the train set. Every experiment regards changing the values of one parameter. The concept uses the validation data to calculate the accuracy score and record it for every epoch. After completing the training for every value of the parameter, a comparison of the achieved maximum accuracy for every value demonstrates the Bidirectional extension's impact on maximum accuracy.

An update of the parameters from one experiment to the other gradually optimizes accuracy, performing and proposing a parameter tuning for both LSTM and BLSTM. The evaluation of the results proposes ways to handle parameters to utilize the Bidirectional extension of LSTM.

4. Results.

4.1 Batch Size.

The study's first experiment aimed to demonstrate how the bidirectional extension could influence the optimal batch size. There were 6 sizes tested: [16, 32, 64, 128, 256, 512]. Figure 6 visualizes the maximum accuracies achieved for every batch size. LSTM and BLSTM seem to provide higher

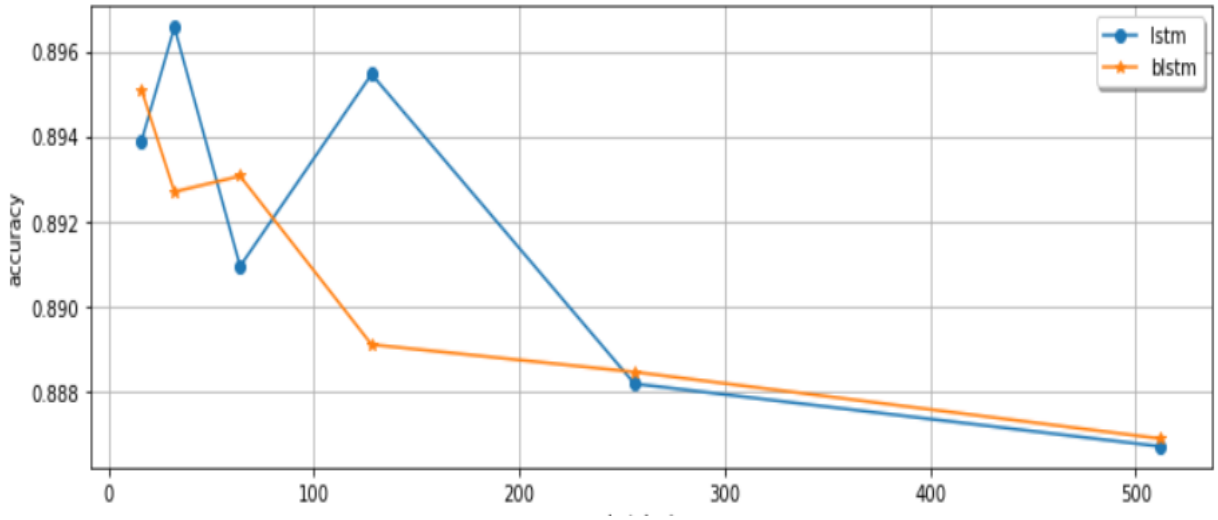


Figure 6. Validation Accuracy VS Batch Size.

accuracies for smaller batch sizes. BLSTM seems to be more stable than LSTM despite achieving lower accuracies. The highest LSTM accuracy was0.8966 for batch size 32 versus 0.8951 for batch size 16 of BLSTM.

4.2 Dropout after Embedding Layer.

The second experiment analyzed the dropout after the embedding layer. The embedding layer provides 64 dimensions and four (4) dropout values were tested: [0.10, 0.25, 0.50, 0.75]. The batch size for this iteration was set to 64 for both LSTM and BLSTM. The maximum accuracy for LSTM reached 0.8977 for a 0.50 dropout and BLSTM 0.8960 for a 0.25 dropout. It seems that a dropout rate after embeddings

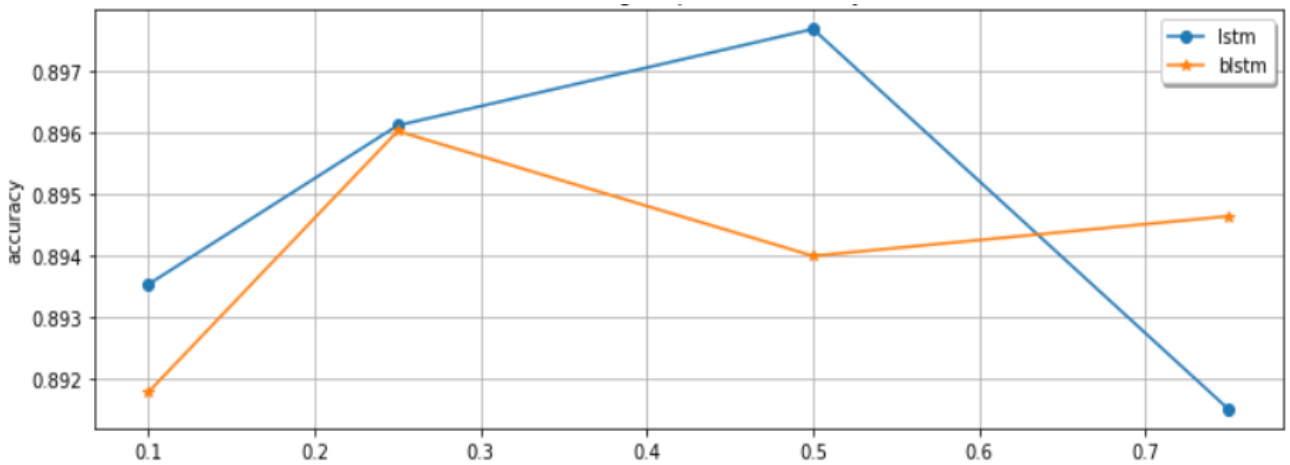


Figure 7. Validation Accuracy VS Dropout Rate after Embedding Layer.

can be useful for both LSTM and BLSTM. BLSTM does not seem to perform better than LSTM, yet.

Figure 7 presents these results graphically, and one may also observe that BLSTM is again more stable than LSTM.

4.3 Dropout after Convolution Layer.

The third experiment analyzed the dropout after the convolution and the max-pooling layer. The dropout values tested were the: [0, 0.10, 0.25, 0.375, 0.50, 0.625, 0.750]. The batch size and the dropout after embedding were set to 64 and 0.50, respectively, for LSTM and BLSTM. The best result for LSTM was a maximum accuracy of 0.8967 achieved for a 0.10 dropout. On the contrary, BLSTM achieved a 0.9008 accuracy for a 0.25 dropout. The visualization of the results (Figure 8) suggests; tuning the dropout rate

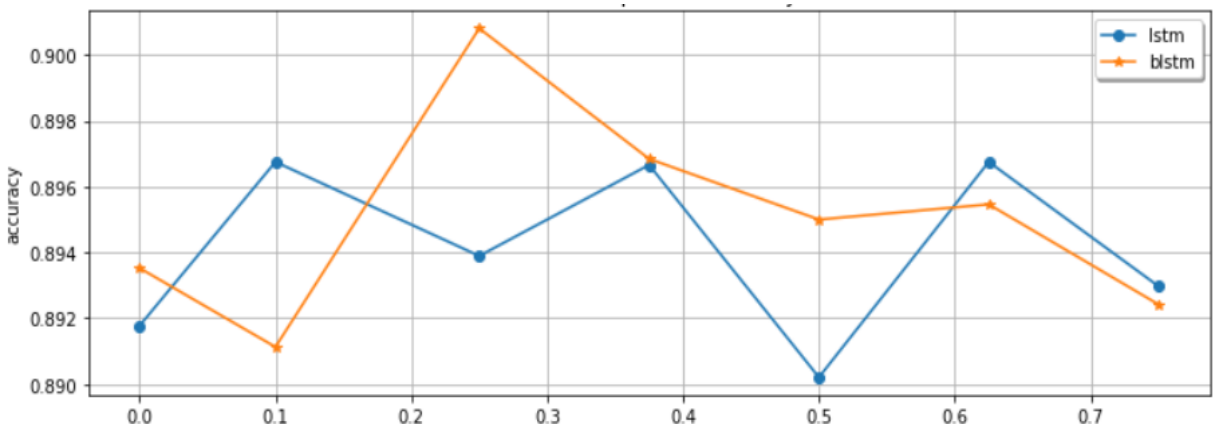


Figure 8. Validation Accuracy VS Dropout Rate after Convolution Layer

after convolution and max-pooling may prove more important when extending LSTM to Bidirectional.

4.4 LSTM Internal Non-Recurrent Dropout.

The LSTM layer provides two internal dropout parameters. A recurrent dropout and a non-recurrent dropout. Internal Non-Recurrent Dropout refers to the fraction of the units to drop for the inputs' linear transformation. The fourth experiment of this study tested how Bidirectional extension may influence maximum accuracy in a flow of different dropout rates than a simple LSTM after performing some parameter searching. The batch size and the dropout after embedding did not change. Dropout after the convolution layer is set to 0.10 for LSTM and 0.25 for BLSTM. The experiment resulted in LSTM

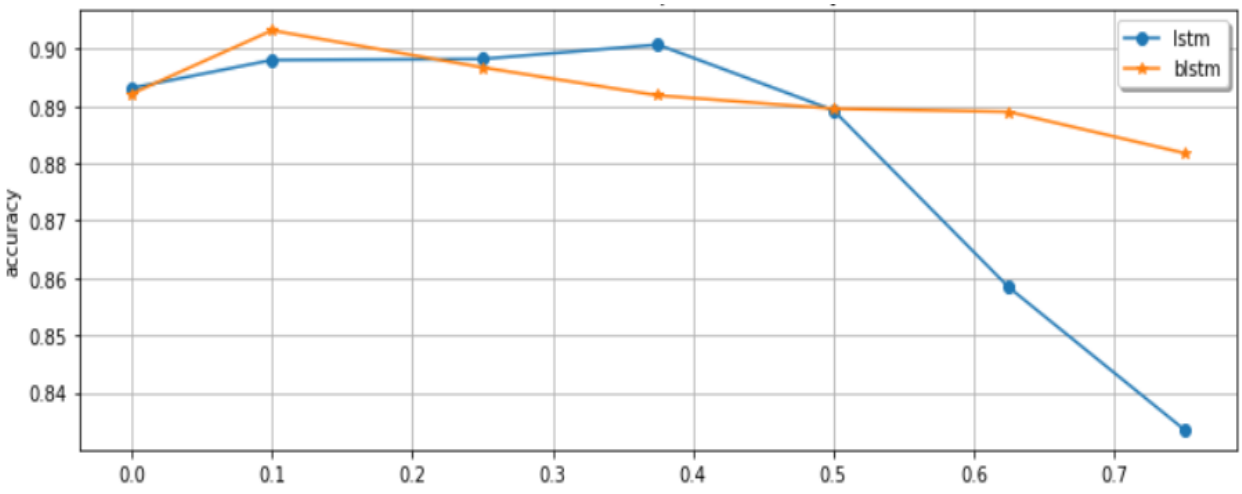


Figure 9. Validation Accuracy VS LSTM Internal Dropout

achieving a 0.9007 maximum accuracy for a 0.375 dropout rate and BLSTM a 0.9032 for a 0.10 dropout

rate. Both LSTM and BLSTM appeared to earn a bit of accuracy, and previous observation about BLSTM being more stable but demanding careful parameter tuning was also confirmed.

4.5 LSTM Recurrent Dropout.

In this study, recurrent dropout is the fraction of the units to drop for the linear transformation of the

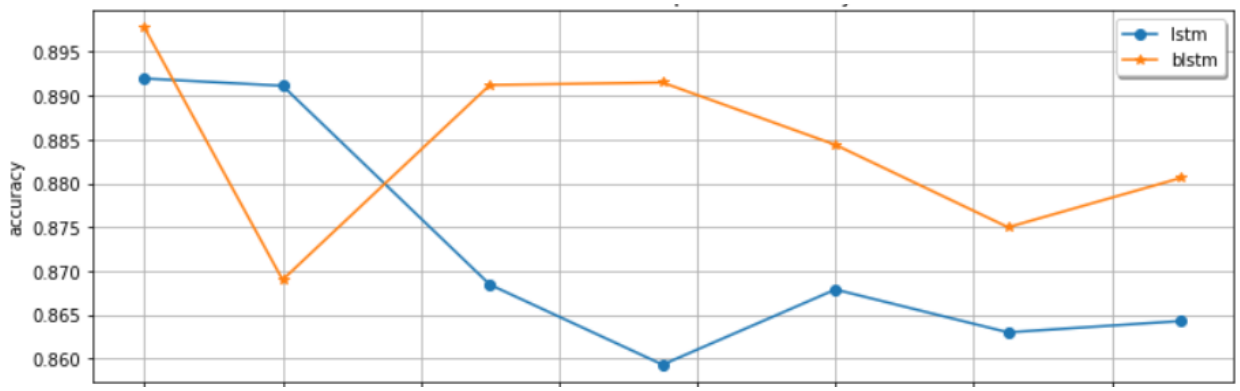
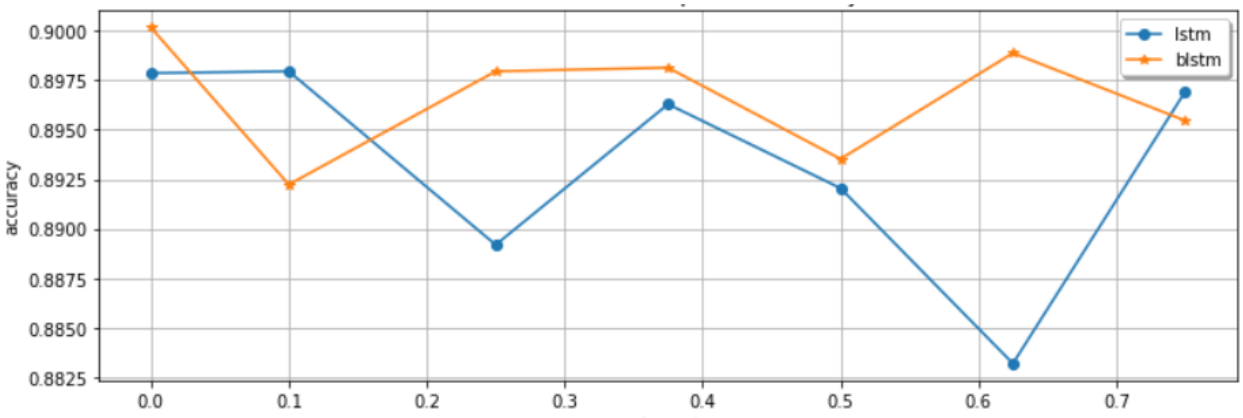


Figure 10. Validation Accuracy VS Recurrent Dropout Rate

recurrent state. The fifth experiment explored the possibility of improving accuracy when training with several recurrent dropout rates. The rates to run are: [0, 0.10, 0.25, 0.375, 0.50, 0.625, 0.75] while the parameters of non-recurrent dropout were updated to 0.375 for LSTM and 0.10 for BLSTM. This experiment indicated that recurrent dropout in the LSTM cell limits the amount of information the model receives. Nevertheless, BLSTM accuracy seems steadily higher than the LSTM.

4.6 Final Dropout Layer

The previous experiment suggested that recurrent dropout in the LSTM cell cannot further improve the accuracies nor for LSTM neither for BLSTM. For validating our results until this stage, a final experiment ran with dropout rates: [0, 0.10, 0.25, 0.375, 0.50, 0.625, 0.75], expecting that a non-zero rate could not beat previous highest accuracies. A different result would mean that dropout rates could further receive updates with greater values. The highest accuracy for BLSTM was 0.9002, achieved for a zero rate, while the highest accuracy for LSTM was 0.8980, achieved with a 0.10 rate.



5. Future Work

The contribution of DL and Text Preprocessing in achieving a classification accuracy greater than 0.90 could support additional research. In the Text Preprocessing section, we introduced the abstraction token. However, it was out of this study's scope to analyze it further and discover its possible contribution to the accuracy. We strongly believe that finding ways to increase abstraction features in a model may utilize new entities. We wish to measure the abstraction's token contribution to sentiment analysis accuracy and extend it to lexical entities different than stopwords.

This study was not a data science contest. We achieved a good classification accuracy while analyzing the difference of the Bidirectional extension to LSTM. Because of this, we did not later Embedding or LSTM units, nor we performed an exhaustive grid search of parameters. Another research may analyze the dropout rates in a flow of LSTM or Embedding units.

6. Conclusions

In this study, about 32,000 texts were downloaded, processed, and annotated for sentiment polarity. Our purpose was to use a labeled dataset to analyze the impact of the Bidirectional extension of the LSTM layer in a DL classifier in the context of Text Classification for Sentiment analysis. We conducted five experiments to analyze the connection of dropout rates to the maximum accuracy the classifier can achieve and one to analyze the batch size.

The findings may support the notion that BLSTM can outperform LSTM when tuned carefully and with detail. The experiments also demonstrated that while BLSTM needs meticulous tuning to maximize its predictability, it also can handle more successfully than LSTM inaccurate and unlucky parameter tuning. We also indicated that LSTM efficiency could vary, and randomness may play should be considered when training a model.

We also analyzed the idea of combining Dropout layers after Embeddings, Convolution, and LSTM layers, and we found no proof against this. The same idea also suggests that the final Dropout layer may be of no use if the previous ones are combined successfully. The first experiment (4.1) also confirmed the idea that a smaller batch size can achieve higher accuracy.

We proved that sentiment analysis using DL could be successful for text pieces characterized by formality and objectivity like the ones coming from prominent organizations like CNN or FOX with the primary purpose to inform rather than express personal and subjective opinions and judgments.

References

Aurélien Géron.(2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems

Borna, Keivan & Ghanbari, Reza.(2019. Hierarchical LSTM network for text classification. SN Applied Sciences. 1. 1124. 10.1007/s42452-019-1165-1.

Elghannam, F. (2019). 'Text representation and classification based on bi-gram alphabet '. Journal of King Saud University - Computer and Information Sciences. https://doi.org/10.1016/j.jksuci.2019.01.005

Han, E.-H.S., Karypis, G. and Kumar, V. (2001). 'Text categorization using weight-adjusted k nearest neighbor classification'. Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science), 2035, pp. 53-65.

Hassan and A. Mahmood. (2017). "Efficient Deep Learning Model for Text Classification Based on Recurrent and Convolutional Layers," 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), Cancun, 2017, pp. 1108-1113, doi: 10.1109/ICMLA.2017.00009.

Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.

Hu, X. and Liu, H. (2012). 'Text analytics in social media'. Mining Text Data, 9781461432234, pp. 385-414. https://doi.org/10.1007/978-1-4614-3223-4_12

J.R. Firth. (1957), Papers in Linguistics 1934–51, October 2007, International Journal of Applied Linguistics 17(3):402 – 413, DOI: 10.1111/j.1473-4192.2007.00164.x

Jiazhu Dai, Chuanshuai Chen, YuFeng Li (2020) School of Computer Engineering and Technology, Shanghai University, Shanghai, China DOI 10.1109/ACCESS.2019.2941376, IEEE Access

Leopold, E. and Kindermann, J. (2002). 'Text categorization with support vector machines. How to represent texts in input space?'. Machine Learning, 46 (1-3), art. no. 380516, pp. 423-444.

László Nemes & Attila Kiss (2020): Social media sentiment analysis based on COVID-19, Journal of Information and Telecommunication, DOI: 10.1080/24751839.2020.1790793

Mike Schuster & Kuldip K. Paliwal. (1997): Bidirectional Recurrent Neural Networks, IEEE TRANSACTIONS ON SIGNAL PROCESSING, VOL. 45, NO. 11, NOVEMBER 1997 2673

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. (2014). Dropout: A simple way to prevent neural networks from overfitting. In Journal of Machine Learning Research 15, pages 1929–1958, 2014.

Rao, Adithya & Spasojevic, Nemanja. (2016). Actionable and Political Text Classification using Word Embeddings and LSTM.

Shah, F.P. and Patel, V. (2016). 'A Review on Feature Selection and Featur e Extraction for Text Classification'. Proceedings of the 2016 IEEE Intern ational Conference on Wireless 30 Communications, Signal Processing and Ne tworking, WiSPNET 2016, art. no. 7566545, pp. 2264-2268

S. Ruder. (2017). An overview of gradient descent optimization algorithms. arXiv:1609.04747

T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed re presentations of words and phrases and their compositionality. In Proceedi ngs of NIPS, 2013, 2013.

Y. Kim. Convolutional neural networks for sentence classification. Proceed ings of the 2014 Conference on Empirical Methods in Natural Language Proce ssing (EMNLP), page 1746aĂŞ1751, 2014.