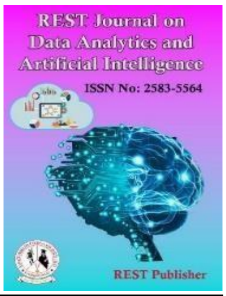
REST Journal on Data Analytics and Artificial Intelligence
Vol: 3(3), September 2024
REST Publisher; ISSN: 2583-5564
Website: http: //restpublisher.com/journals/jdaai/
DOI: https://doi.org/10.46632/jdaai/3/3/11



Twitter Sentiment Analysis with LSTM Neural Networks

*Jayanth Kande
Southern University and A&M College, Baton Rouge, Louisiana, United States.
**Corresponding author: jayanth.m1229@gmail.com*

Abstract: *This project delves into sentiment analysis on Twitter using Long Short-Term Memory (LSTM) Neural Networks in conjunction with Global Vectors for Word Representation (GloVe). The study explores the properties of tweets, preprocessing steps, and applying GloVe embedding's to map words to vectors. The classifier's design and training parameters are detailed, and the results are compared with baselines, revealing the LSTM's superiority in handling sequential language data. Furthermore, trials explore how changing the quantity of fully connected layers and LSTM time steps affects accuracy. The findings suggest the importance of sequential processing in natural language processing (NLP) tasks. However, accuracy may benefit from flexible LSTM time steps aligned with tweet length. This project highlights the potential of LSTM neural networks for sentiment analysis on social media platforms like Twitter.*

1. INTRODUCTION

With the development of Profound Learning, Regular Lan- guage Handling (NLP) has turned into a hotly debated issue of interest and exploration in AI. The essential objective of NLP is to take advantage of regular language messages or discourse with PC applications to accomplish something material to my day-to-day routine. Joined with computational phonetic and discourse innovation, NLP is unendingly being utilized as a significant part of Human Language Advances, which plans to create and carry out proper devices for PC frameworks to figure out normal dialects and to execute wanted undertakings. Feeling examination centers around grasping how feelings, assessments, and opinions of individuals are communicated in texts. Opinion mining isn't just a single piece of the review for Human Language Innovations; in addition, it is one of the most dynamic exploration subjects of NLP. As informal organizations are vigorously utilized by us, e.g., Twitter, to offer viewpoints and feelings, the need to use progressed inves- tigation of opinion examination begins to emerge, particularly for business benefits [1]. Throughout this project, I use *LSTM* with *GloVe* for opinion mining. LSTM neural networks have been used successfully in several sequential-based tasks such as speech recognition [2]. Video action recognition [3], and GloVe is considered better than Skip-Gram Model [4]. The rest of this report is organized as follows. In section II, I observe the tweets to reveal their properties. In section III, I list what I do in preprocessing. In section IV, I introduce GloVe embedding, which I use to map words to vectors. In section V, I describe the design of my classifier. I specify training parameters in section VI and present the results of my experiments. Lastly, I propose possible ways to increase accuracy in section VII.

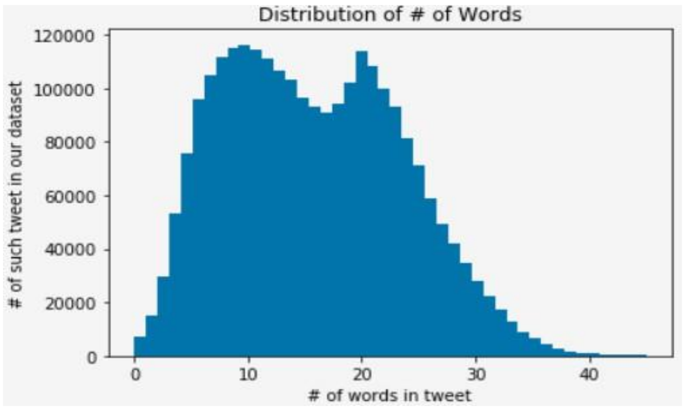


FIGURE 1. Distribution of # of words

2. DATASET ANALYSIS

I utilize a substantial Twitter dataset containing tweets numbered 2,500,000, divided evenly between 1,250,000 and 1,250,000 positive and negative tweets, respectively. Every post holds 140 characters and normally has either a positive or negative grin. Moreover, most tweets contain under 45 words (Fig. 1). Moreover, I see that the language of tweets changes, for the most part, to English for certain dialects like French, Malay, etc. I describe the accuracy of my model for analyzing opinions using the Twitter test dataset, which consists of 10,000 unlabeled tweets.

3. DATA PREPARATION

Before continuing with the preparation step, I removed few good-for-nothing words and replaced them with additional valuable terms by preprocessing the words. I will justify why I made the changes and chose to utilize explicit words instead in segment IV.

A. Words and Characters Removal: In view of an exploratory examination of the preparation dataset, I noticed a few trivial words and qualities that gave practically no data to opinion investigation. Subsequently, I eliminate:

- **Numbers**

Any number used as a word, for example, the timestamps 23:55 and 1:20, is essential to group feelings.

- **Repetitive Characters**

I frequently see something like *'excited* or *'thanks* in tweets. I removed any that were repeated characters so that during the GloVe insertion process they will be considered as *'excited'* and *'thanks'*.

B. Words Transformation: I also eliminated a few words and characters and removed a few parts in tweets to give more significant portrayals. In this way, we:

- **Expand English Contraction**

Compressions are frequently used in created English, for instance," I'm," "You're dead serious," and others. To handle the words provided by GloVe, I elongate the words to" I am," "You are," "He is not," and so on.

- **Lower Repetitive Punctuations**

I used multiple punctuation in the tweets, e.g.,'!!!'.

- **Highlight Sentiment Words**

In English, certain words have unambiguous opinions. For example, "*convenient*" is in all likelihood certain while "*abuse*" has a negative tendency. To accomplish this, I use evaluation lexicons such as dataset [5], which holds a list of positive and negative words, and I prepend the labels as "*positive*" or "*negative* accordingly.

Split Hashtags (#) into Words

Hashtags are primarily applied in tweets and highlight the tweets' significance. Whatever may be the case, when you have more than one word behind a hashtag, it can be confusing. For example, you can get both the meanings

- "*meaningless*" or" meaning *less*" from *"#meaningless*." To address the issue, I utilized a word dictionary sourced from a subset obtained from Wikipedia. Through ad- vanced programming, I parsed the hashtag and then mapped the words to vectors using the GloVe framework.

- **Emojis to Special Words**

When you look into the mouth of an emoji, the emo- tion it represents is easily understood. For example,':)' insinuates commonly to a positive explanation, and':(' interfaces with a negative declaration. Therefore, I switched few emojis into a couple of exceptional words, like'*<lolface>'* and'*<sadface>'*. Emotions behind the emoji-like': - o' is deniable, so I employ'*<neutralface>'* for the present scenario. There arises a need to change a few emojis into unprecedented words when using the GloVe embedding dataset from the Standford NLP bundle, as not all are open in the dataset.

4. GLOVE EMBEDDING

A computation for arranging words to vectors is the Overall Vectors for Word Depiction (GloVe) [6]. Speculation of the computation relies upon co-occasion bits of knowledge. For instance, the co-occasion probability of 'ice' 'areas of strength for and' higher than that of 'ice' and 'gas.' I let the system read a couple of articles, measure the times that two words co-occur, and use a subgradient plunge to restrict the going with capacity:

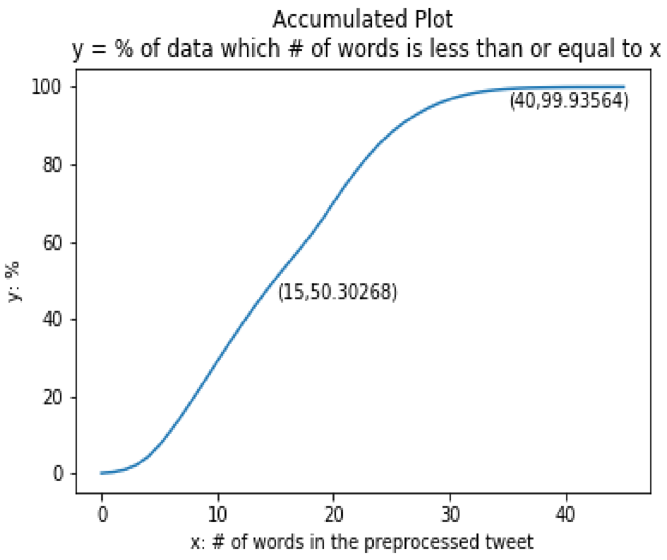


FIGURE 2. Summarized Plot of Hashtag Words.

$$J = \sum_{i,j=1} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

here the weighting ability is *f*, *V* is the size of language, X_{ij} is co-occasion probability, while *w* and *b* are limits. I change and highlight words to feeling words in my preprocessing to use the pre-arranged word vectors for tweets from Stanford NLP pack [7]. The GloVe list integrates each changed word, for instance, 'positive,' <sadface>,' and addresses every standard word with a 200-perspective vector. With the vectors that have been arranged beforehand, I produce a 40x200 framework for every tweet as a commitment to the LSTM:

$$\begin{bmatrix} \text{word vector of 1st word} \\ \text{word vector of 2nd word} \\ \ldots \\ \text{word vector of 39th word} \\ \text{word vector of 40th word} \end{bmatrix}$$

Suppose the words do not exceed 40; I pad them with zeros. The choice of 40 columns is motivated by two reasons: firstly, the distribution of words in 99.93% of preprocessed tweets (Fig. 2) shows that most tweets do not exceed 40 words; secondly, this choice enhances precision. (Table III)

4. ARCHITECTURE

Figure 3 shows the overall design utilized in this analysis. A 200-layered GloVe inserted word is added to the LSTM [8], leading to it updating its boundaries by utilizing the accompanying repeat condition. This occurs ar each time step:

$$\text{input gate } i_t = sigm\,(W_i \cdot [x_t, h_{t-1}] + b_i)$$
$$\text{forget gate } f_t = sigm\,(W_f \cdot [x_t, h_{t-1}] + b_f)$$
$$\text{output gate } o_t = sigm\,(W_o \cdot [x_t, h_{t-1}] + b_o)$$
$$\text{input modulation gate } g_t = tanh(W_g \cdot [x_t, h_{t-1}] + b_g)$$

$$\text{memory unit } c_t = f_t \circ c_{t-1} + i_t \circ g_t$$
$$\text{hidden unit } h_t = o_t \circ tanh(c_t)$$

Naturally, using sigmoid functions in the gates enables LSTMs to manage the data flow through the unit effectively, taking into account both the current data and information from previous timesteps. This gating mechanism allows LSTMs to decide what information to retain or forget, making them adept at learning long-term dependencies. This capability is particularly crucial because vanilla Recurrent Neural Networks (RNNs) often struggle with the vanishing/exploding gradient problem [9], [10], which hampers their ability to learn long-range dependencies.

I opted for LSTMs in my implementation due to their robustness in handling these long-term dependencies, which are essential for understanding the context in sequences like tweets. Specifically, for tweets with fewer than 40 words, I padded the sequence with null vectors for the remaining timesteps to maintain a consistent input length. This padding ensures that the LSTM processes each tweet as a fixed- length sequence, simplifying batch processing and maintaining uniformity in the input data.

The LSTM processes these sequences and outputs a result at t=40, representing the padded sequence's final timestep. This output is then passed through four connected layers with [512, 512, 512, 2] units, respectively. These fully connected layers are responsible for transforming the LSTM's output into a suitable form for classification. Each layer applies a set of learned weights to its inputs, enabling the network to capture complex patterns in the data.

The output from the last fully connected layer is processed using a sigmoid activation function. The sigmoid function is particularly suitable for binary classification tasks because it squashes the output to a range between 0 and 1, allowing it to represent the probability of each class. In this context, the two units in the final layer likely correspond to the two classes in the sentiment analysis task (e.g., positive and negative sentiments).
The use of LSTMs combined with fully connected layers and sigmoid activation allows for a robust and effective model for tweet sentiment analysis. The LSTM's ability to capture long-term dependencies and the structured processing of outputs through dense layers ensure that the model can accurately classify sentiments by considering the sequential nature of the input data.

5. EXPERIMENTS

A. Training: I set up the model parameters using the Glorot uniform initializer [11], which helps initialize the weights to maintain the variance of the activations throughout the layers. I employed stochastic gradient descent with a batch size of 1000 for optimization to improve convergence. Additionally, I applied the RMSProp optimization algorithm [12], with a base learning rate of 5e-4, a decay rate of 0.9, and no momentum This optimizer choice helps handle the adaptive learning rates effectively, especially for the types of tasks involving non- stationary targets.
The LSTM was configured with 1024 units, unrolled for 40 timesteps. The model was trained for ten epochs, a duration determined to balance between performance and overfitting. The dataset was split into training and validation sets with a 90:10 ratios, ensuring that the model has ample data to learn from while retaining a portion for evaluating its performance. To prevent overfitting, I applied dropout [13] to all fully connected layers with a keep probability of 0.5. This regular- ization technique randomly drops units during training, forcing the network to generalize better. Additionally, I converted the binary labels to one-hot encoding to facilitate multi-class classification, although the problem at hand involves two
classes.

The experiments were conducted on a machine equipped with two NVIDIA Titan X GPUs, significantly speeding up the training process, which took approximately 4 hours. I utilized the TensorFlow library [14] for developing and training the model, leveraging its robust features for building and optimiz- ing deep learning models.

In summary, the combination of the Glorot initializer, RM- SProp optimizer, and dropout regularization, along with one- hot label encoding and leveraging powerful GPU hardware, ensured efficient and effective training of the LSTM model for Twitter sentiment analysis. The careful setup and resource utilization were pivotal in achieving a model that generalizes well without overfitting.

B. Results: During my primary trials, I evaluated the engineering ap- proach described above against several baseline models: Naive Bayes and a Decision Tree, both implemented using the scikit- learn library [15] with default parameters. Each of these algo- rithms required transforming the sentences into feature vectors. To achieve this, I flattened the GloVe-embedded representation of each word and concatenated these word vectors, forming a vector of length 8000.
The LSTM model significantly outperformed these baseline models, as shown in Table? The superior performance of the LSTM can be attributed to its ability to capture and leverage the sequential dependencies in the input data, which is a fundamental aspect of natural language processing (NLP). In contrast, Naive Bayes and Decision Tree algorithms treat each feature independently, failing to account for the contextual relationships between words. Their outputs are essentially non- linear combinations of all input features without considering the order or structure inherent in the language.

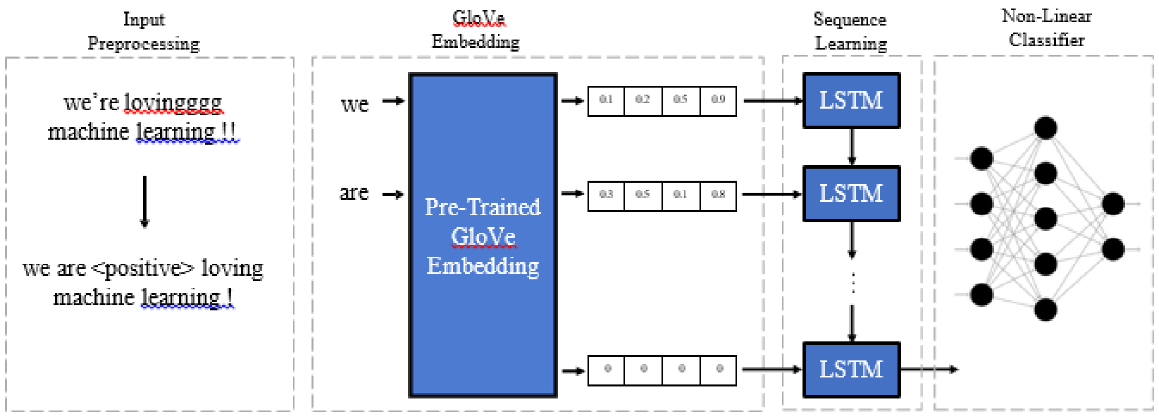


FIGURE 3. Twitter Sentiment Analysis Pipeline

This difference in performance underscores the necessity of using sequential processing techniques like LSTMs for NLP tasks. While traditional machine learning algorithms can be effective for certain types of data, they are not well-suited to handle the complexities of natural language, where the meaning of a sentence is highly dependent on the order and structure of the words. By leveraging the ability of LSTMs to maintain context over sequences of data, we can achieve much higher accuracy in tasks such as sentiment analysis, where understanding the nuance of word sequences is crucial.

The results demonstrate that LSTMs are far superior for tasks involving sequential data compared to models that do not consider language's inherent structure. This highlights the importance of choosing appropriate models to capture the dependencies and contextual information necessary for accurate natural language processing.

In the subsequent examination, I concentrate on the impact of various setups of my engineering on precision. In the following exam, the different setups of my engineering will be my focus. Initially, I fixed the LSTM time steps at 40 and moved the number of associated layers. Each layer was fixed to have 512 units, the last order layer having two units. This was done to avoid exploring different avenues. The outcomes are summed up in table II. Strangely, the main improvement should be visible when incrementing the number of layers from 3 to 4. This will seem OK, assuming I view the result of the LSTM as a component vector addressing the opinion of the Twitter post that isn't straightly distinguishable. In this way, the expressive force of the completely associated layers would increment with profundity. At long last, I can likewise see that the rising quantity of layers in the past 4 brought about a debasement of precision.

Then, I fix the completely associated layers at four layers with [512,512,512,2] units and fluctuate the LSTM timesteps. The outcomes have been summed up in table III. It tends to be seen that diminishing the number of time steps under 40 results in a reduction in execution. This is on the grounds that the feeling of certain tweets must be found towards the finish of the sentence. It is additionally intriguing to take note that raising the number of time steps to 45 significantly diminishes execution. I recorded the exactness swaying around 65% over the 10 ages. This gives me reason to be doubtful whether the number of time steps has been caught in the neighborhood minima during early preparation.

TABLE 1. Validation Performance Comparison

Procedure	Accuracies %
Naive Bayes	62.10
Decision Trees	67.70
LSTM	88.03

TABLE II. Approval exactnesses with differed numbers of completely associated layers. Each layer, except the last order layer, holds units numbering to 512. Those designs with just one layer, in this way, have two units.

FC Layer	Accuracies %
1	87.99
2	88.11
3	88.32
4	91.02
5	89.03

Finally, I show the approval exactness plots (figure **??**) for my subsequent examination, and I see no obvious overfitting indicator.

6. CONCLUSION

In my project on Twitter opinion analysis using Long Short- Term Memory (LSTM) networks, I discovered several key challenges and insights regarding tweet preprocessing and model performance. One of my main difficulties was dealing with the various informal and abbreviated forms of language commonly used on Twitter. For instance, abbreviations like ’idk’ mean ’I don’t have the foggiest idea.’ While my model attempted to handle such variations, I did not fully explore the impact of these linguistic nuances in my experiments. Improving the handling of such informal language could potentially enhance model performance.

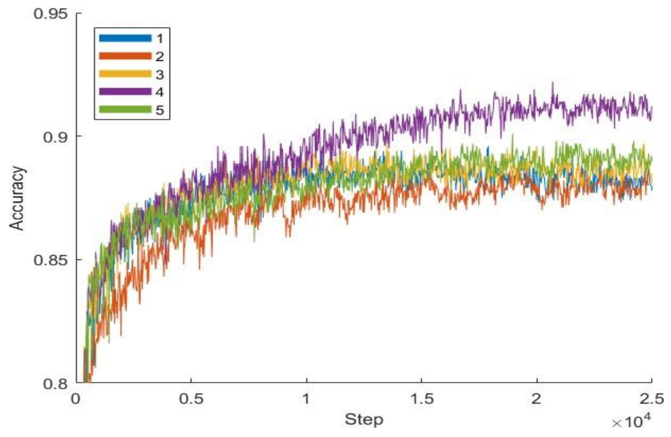


FIGURE 4. Approval precision with changed number of completely associated layers

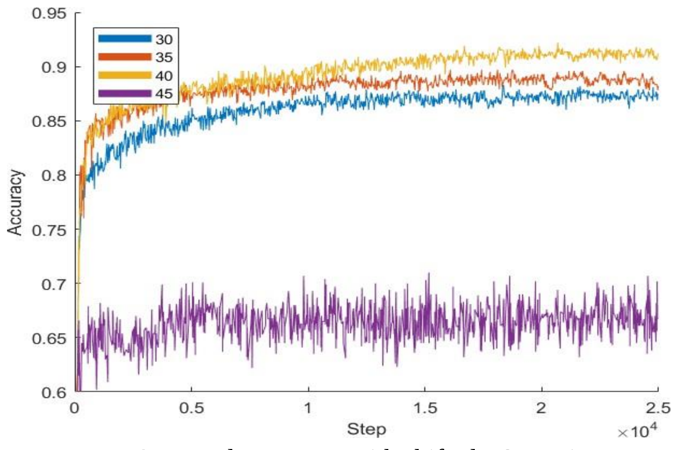


FIGURE 5. Approval exactness with shifted LSTM time steps

Figure 4 and figure 5 Approval precision plots (a) fluctuated quantities for completely associated layers and (b) differed LSTM time steps.

TABLE 3. Different LSTM timesteps with the validation accuracies. All models have four fully connected layers.

Timesteps	Accuracy %
45	67.51
40	91.02
35	88.16
30	87.35

Additionally, I did not investigate the syntactical structure of sentences in my analysis. This oversight became apparent when I realized that simply linking a word like ’hate’ to a negative sentiment could be misleading. For example, a tweet stating ’I don’t despise’ conveys a positive sentiment despite containing the word ’despise.’ Correctly parsing and understanding sentence structure could significantly improve sentiment analysis accuracy.

Another realization was the importance of providing the LSTM with sufficient context regarding the number of words. Initially, my model treated each tweet with a fixed timestep, which might not be optimal given the varying lengths of tweets. Allowing the LSTM timesteps to adjust based on the word count of each tweet dynamically could potentially improve precision. This approach would enable the model better to capture the context and nuances of each tweet, leading to a more accurate sentiment analysis.

In conclusion, addressing the challenges of informal language, understanding sentence structure, and dynamically adjusting LSTM time steps are crucial steps that could enhance the performance of Twitter opinion analysis models. These improvements would enable the model to capture better the complexity and variability inherent in social media data.

REFERENCES

[1]. P. P. A Pak, “Twitter as a corpus for sentiment analysis and opinion mining,” LREC, 2010.

[2]. N. J. A Graves, “Towards end-to-end speech recognition with recurrent neural networks,” Proceedings of Machine Learning Research, 2014.

[3]. D. et al, “Long-term recurrent convolutional networks for visual recogni- tion and description,” Computer Vision and Pattern Recognition (CVPR), 2014.

[4]. T. M. et al, “Distributed representations of words and phrases and their compositionality,” NIPS, 2013.

[5]. M. Hu and B. Liu, “Mining and summarizing customer reviews,” Pro- ceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2004), 2004.

[6]. R. S. Jeffrey Pennington and C. D. Manning, “Glove: Global vectors for word representation,” Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014.

[7]. R. S. Stanford NLP group, Jeffrey Pennington and C. D. Manning, “glove.twitter.27b.200d.txt,” https://nlp.stanford.edu/projects/glove/.

[8]. S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural computation, 1997.

[9]. Y. Bengio and P. F. Patrice Simard and, “Learning long-term dependen- cies with gradient descent is difficult,” IEEE Transactions on Neural Networks, 1994.

[10]. S. Hochreiter, Y. Bengio, and J. S. Paolo Frasconi and, “Gradient flow in recurrent nets: The difficulty of learning long-term dependencies.” A field guide to dynamical recurrent neural networks, 2001.

[11]. Y. B. Xavier Glorot and, “Understanding the difficulty of training deep feedforward neural networks,” Proceedings of Machine Learning Research, 2010.

[12]. T. Tieleman and G. E. Hinton, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,” COURSERA: Neural Networks for Machine Learning, 2012.

[13]. N. Srivastava, G. Hinton, A. Krizhevsky, and R. S. Ilya Sutskever and, “Dropout: A simple way to prevent neural networks from overfitting,” Journal of Machine Learning Research, 2014.

[14]. M. Abadi et al., “Tensorflow: Large-scale machine learning on hetero- geneous distributed systems,” LREC, 2010.

[15]. L. Buitinck et al., “API design for machine learning software: experiences from the scikit-learn project,” ECML PKDD Workshop: Languages for Data Mining and Machine Learning, 2013.