# Sentiment Analysis Using RNN, LSTM, GRU, Bidirectional RNN

Saurabh Roy

***Abstract*— With the rapid development of the Internet, more and more users expressed their views on the Internet. Therefore, the big data of texts are generated on the Internet. Social media is generating a vast amount of sentiment rich data in the form of tweets, status updates, blog posts etc. In the era of big data and deep learning, mining the sentiment tendencies contained in massive texts on the Internet through natural language processing technology has become an important way of public opinion supervision. This analysis has a vast and great applications such as stock price prediction, election result prediction, and so on. In this paper, I have applied a simple and efficient Neural Language Model approach for text classification (Sentiment Analysis) that relies only on unsupervised word representation inputs along with word embedding technique, which has improved the model to capture syntactic and semantic word relationships. This paper shows that applying the RNN, LSTM, GRU, Bidirectional RNN models achieves excellent result on the customer review dataset.**

***Index Terms*—Bidirectional RNN, GRU, LSTM, natural language processing, RNN, sentiment analysis.**

## I. INTRODUCTION

NATURAL language processing (NLP) is all about creating systems that process or "understand" language in order to perform certain tasks. These tasks could include:

- Question Answering – The main job of technologies like Siri, Alexa, and Cortana
- Sentiment Analysis – Determining the emotional tone behind a piece of text
- Image to Text Mappings – Generating a caption for an input image
- Machine Translation – Translating a paragraph of text to another language
- Speech Recognition – Having computers recognize spoken words

Sentiment Analysis is the process of identifying and classifying the ideas expressed in the form of text, to determine whether the writer's emotion towards a particular subject, product, etc. is positive, negative, or neutral. Sentiment analysis has vast scopes including contextual semantic search, brand value detection, stock price prediction, customer care, and so on.

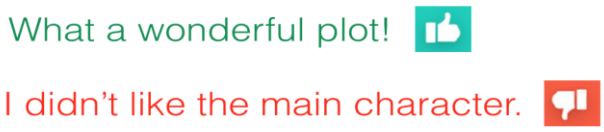


Fig. 1. Example of positive and negative class sentiments.

Today, sentiment analysis has a great trend in the field of marketing. Especially the big vendors like Google, Microsoft, IBM, Amazon, Facebook offer their own sentiment detection services like Google NLP API, Microsoft Azure, IBM tone analyzer, AWS comprehend.

As beautifully explained in reference [1] (page 367-412), I got to know the basic concepts behind the working of Recurrent Neural Network (RNN), vanishing and exploding gradient problem in RNN, Long Short Term Memory (LSTM), and Gated Recurrent Unit (GRU). Reference [2] shows how LSTM works, in depth and describe how it is more powerful than simple RNNs. Reference [3] shows the in-depth working of GRU which is a simpler version of LSTM. Reference [4] shows the in-depth working of bidirectional RNNs.

Reference [5] and reference [6] explains how LSTMs and GRUs works on text data for sentiment analysis by capturing the semantic relationship between different words in the text. Also, I have learned how to apply natural language processing techniques on raw data such as twitter dataset [7]. I learned about different word representations such as word2vec, glove, and word-embeddings which enhances the model accuracy by generating required semantic relationship between the sequence of words in the text [8].

## II. METHODOLOGY AND APPROACH

I have applied four models for sentiment analysis and trained and tested them over the customer review dataset.

- RNN model that employs embedding layer followed by a simple RNN layer followed by a fully connected layer with dropouts and then by an activation layer.
- LSTM model that employs embedding layer followed by an LSTM layer followed by a fully connected layer with dropouts and then by an activation layer.
- GRU model that employs embedding layer followed by a GRU layer followed by a fully connected layer

This journal is submitted for review on April 26, 2020. This work was supported in part by International Institute of Information Technology Naya Raipur.

S. Roy is with department of computer science, International Institute of Information Technology Naya Raipur, Raipur, CG, 493661 India (e-mail: saurabhr17100@iiitnr.edu.in).

with dropouts and then by an activation layer.

- Bidirectional RNN model that employs embedding layer followed by a bidirectional LSTM layer followed by a fully connected layer with dropouts and then by an activation layer.

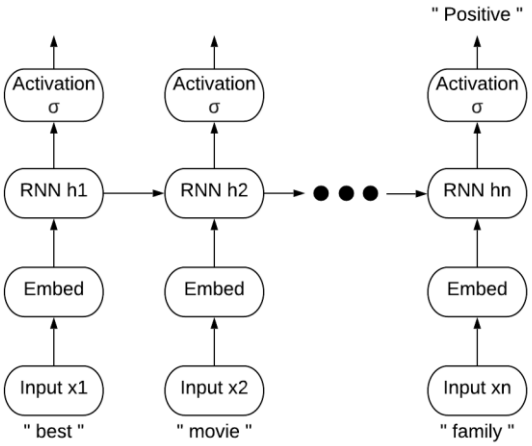


Fig. 2. RNN model architecture.

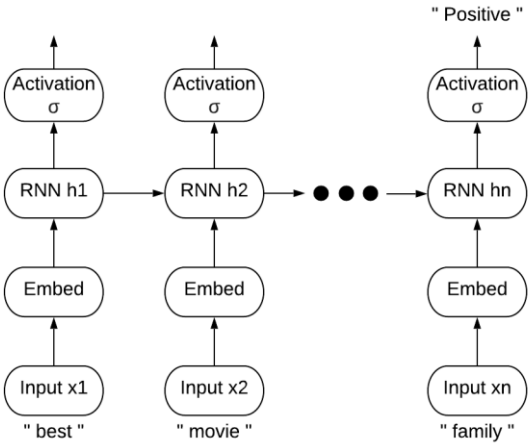


Fig. 3. LSTM model architecture.

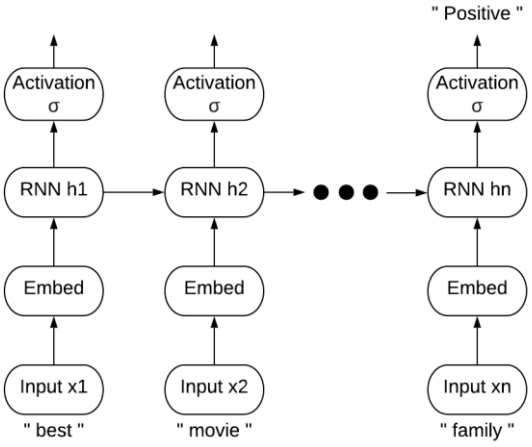


Fig. 4. GRU model architecture.

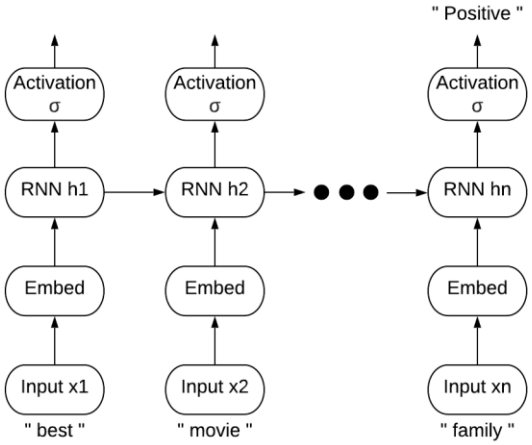


Fig. 5. Bidirectional RNN model architecture.

## A. Dataset

For sentiment analysis, I have combined together two datasets as a single customer review training dataset.

The first one is the IMDB movie review dataset which is publicly available at Keras. This dataset includes 25,000 movies reviews from IMDB, labelled by sentiment (positive/negative).

The second one is hotel review dataset. This dataset includes more than 5 lakh customer reviews and scoring of about 1500 luxury hotels across Europe. This dataset is fetched from booking.com. The dataset looks like Fig. 6.

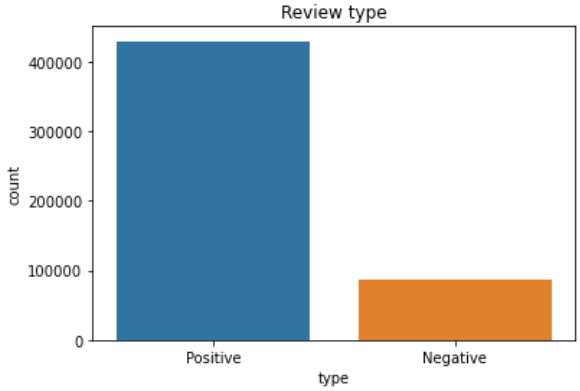


Fig. 6. Distribution of reviews among different classes.

## B. Data Preprocessing

### 1) Balancing Dataset

Since the dataset is more biased to the positive class, so I had randomly dropped the extra positive instances and shuffled the whole training dataset. After balancing, the dataset looks like Fig. 7.

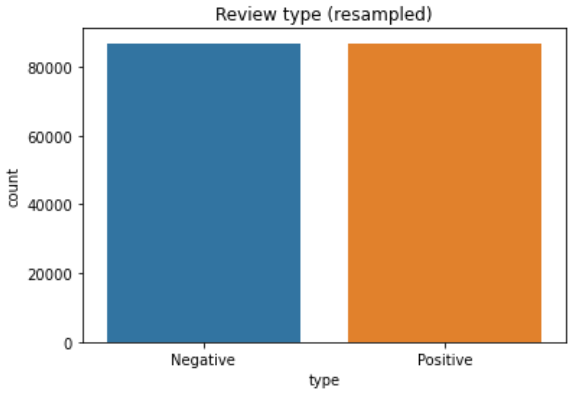


Fig. 7. Distribution of reviews among different classes after balancing the dataset.

### 2) Removing Punctuations and Extracting Words

Punctuations are not needed to build a model. Therefore, all the punctuation symbols are removed from each of the reviews and then each review is converted into a list of words.

### 3) Encoding Reviews

Each of the word in the review list is encoded to an integer index with the help of a dictionary or vocabulary. Dictionary is created from a large text corpus by mapping each unique word with its respective frequency. Here, the words are encoded as indexes which represents overall frequency of that word in the corpus. For example, the integer 5 is the encoded index for the third most frequent word in the data.

### 4) Padding Sequences

The RNN will take sequences of constant length. This length is the is defined to be 200. Since the reviews can be of different lengths, I will trim each review to its first 200 words. If reviews are shorter than 200 words, I will pad them with zeros. That is, if the review is ['best', 'movie', 'ever'], [117, 18, 128] as encodings, the row will look like [0, 0, 0, ..., 0, 117, 18, 128]. These word encodings will be passed on to the RNN model as inputs.

## C. Embedding Layer

The embedding layer will give a meaningful representation of the words by converting each word to a 128-dimensional vector such that the words with similar contextual meaning will be placed nearby each other whereas the words with dissimilar meanings will be placed separately.

For example, the word vectors of the words 'love' and 'adore' will be placed together as they have same contextual meaning as for many cases, they stand for the positive sentiment. Fig. 8 shows word embeddings for the words ['love', 'adore', 'baseball'].

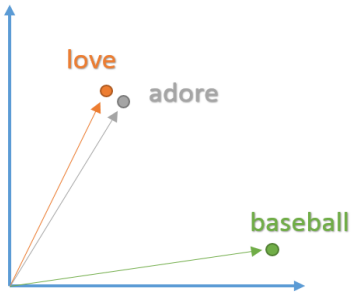


Fig. 8. Word embedding vectors of some words showing similar words are placed together whereas dissimilar words are placed separately.

## D. RNN Layer

The unique aspect of NLP data is that there is a temporal aspect to it. Each word in a sentence depends greatly on what came before and comes after it. In order to account for this dependency, I used a many-to-one recurrent neural network.

They are networks with loops in them, allowing information to persist. In simple words, a recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.

Unfortunately, as the gap between the relevant information and the place that it's needed grows, RNNs become unable to learn to connect the information. This problem occurs due to iteratively decrement of the gradient value and is called vanishing gradient problem.

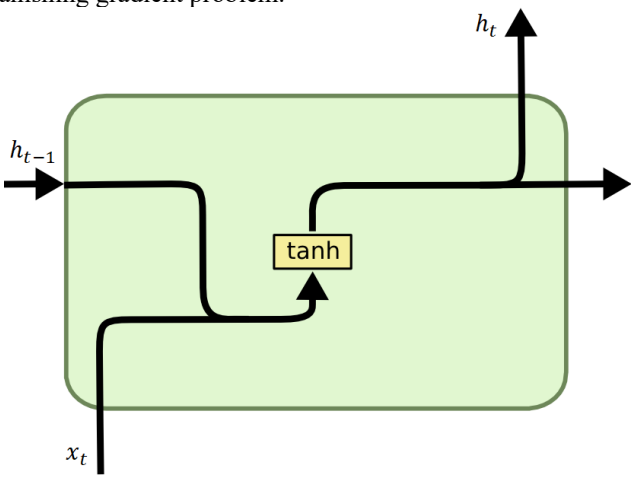


Fig. 9. Structure of an RNN hidden layer at time instant t.

The equations that governs the structure as mentioned in Fig. 9 are:

$$h_t = \tanh(W_H \cdot [h_{t-1}, x_t] + b_H) \quad (1)$$

$$y_t = \sigma(W_Y \cdot h_t + b_Y) \quad (2)$$

Where,

$h_t$ is the output state of the RNN hidden layer at instant t.

$x_t$ is the input to the hidden layer at instant t.

$y_t$ is the final output at instant t, after applying the activation function σ to the hidden layer state $h_t$.

*tanh* function implements a non-linearity that squashes the activations to the range [-1,1].

$W_H$ and $b_H$ represents the combined connection weight and bias matrix between the previous hidden layer state $h_{t-1}$, present input $x_t$, and current hidden layer.

$W_Y$ and $b_Y$ represents the connection weight and bias matrix associated with the hidden layer $h_t$ and the activation layer.

## E. LSTM Layer

LSTMs are a special kind of RNN, capable of learning long-term dependencies. LSTMs are explicitly designed to avoid the vanishing gradient problem. Remembering information for a long period of time is practically their default behavior, not

something they struggle to learn!

LSTMs also have the chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

The key to LSTMs is the cell state. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer outputs numbers between zero and one. A value of zero means "let nothing through," while a value of one means "let everything through!". An LSTM has three of these gates (forget, input, and output gate), to protect and control the cell state.

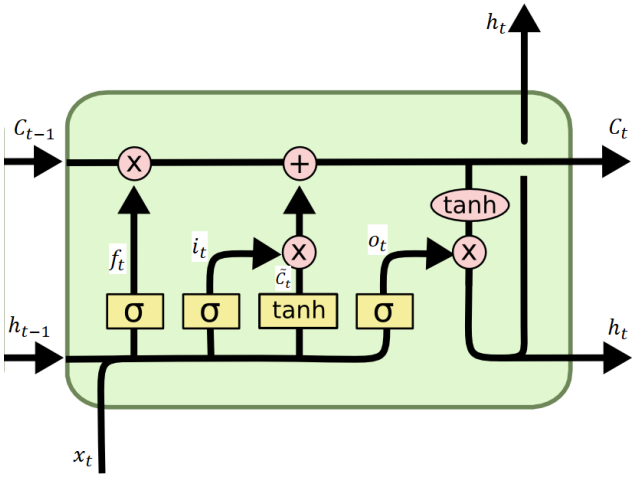


Fig. 10. Structure of an LSTM hidden layer at time instant t.

The equations that governs the structure as mentioned in Fig. 10 are:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{4}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{5}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{6}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{7}$$

$$h_t = o_t * \tanh(C_t) \tag{8}$$

$$y_t = \sigma(W_Y \cdot h_t + b_Y) \tag{9}$$

Where,
$f_t$ is the state of the forget gate at instant t.
$i_t$ is the state of the input gate at instant t.
$o_t$ is the state of the output gate at instant t.
$C_t$ is the cell state at instant t.
$\tilde{C}_t$ is set of new candidates for the cell state at instant t.
$x_t$ is the input to the hidden layer at instant t.
$h_t$ is the output state of the LSTM hidden layer at instant t.
$y_t$ is the final output at instant t, after applying the activation function σ to the hidden layer state $h_t$.
*tanh* function implements a non-linearity that squashes the activations to the range [-1,1].
$W_f$ and $b_f$ represents the combined connection weight and bias matrix between the previous hidden layer state $h_{t-1}$, present input $x_t$, and the forget gate.
$W_i$ and $b_i$ represents the combined connection weight and bias matrix between the previous hidden layer state $h_{t-1}$, present input $x_t$, and the input gate.
$W_o$ and $b_o$ represents the combined connection weight and bias matrix between the previous hidden layer state $h_{t-1}$, present input $x_t$, and the output gate.
$W_C$ and $b_C$ represents the combined connection weight and bias matrix between the previous hidden layer state $h_{t-1}$, present input $x_t$, and the cell state.
$W_Y$ and $b_Y$ represents the connection weight and bias matrix associated with the hidden layer $h_t$ and the activation layer.

### F. GRU Layer

A slightly more dramatic variation on the LSTM is the GRU. It combines the forget and input gates into a single update gate. The output gate is replaced with relevant gate. It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.

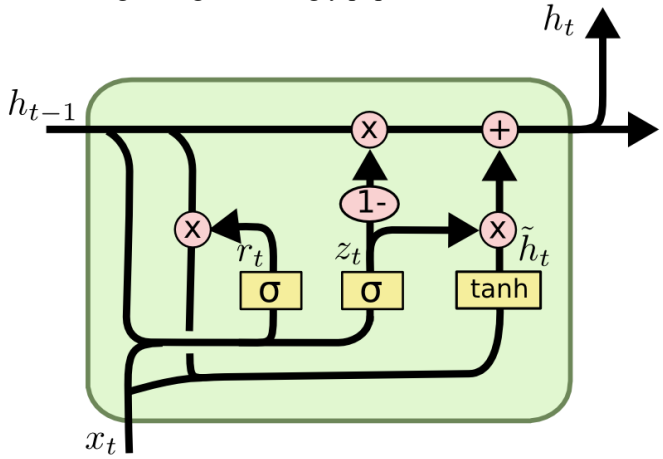


Fig. 11. Structure of a GRU hidden layer at time instant t.

The equations that governs the structure as mentioned in Fig. 11 are:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \tag{10}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \tag{11}$$

$$\tilde{h}_t = \tanh(W_H \cdot [r_t * h_{t-1}, x_t] + b_H) \tag{12}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{13}$$

$$y_t = \sigma(W_Y \cdot h_t + b_Y) \tag{14}$$

Where,
$z_t$ is the state of the update gate at instant t.

$r_t$ is the state of the relevant gate at instant t.
$\hat{h}_t$ is the new candidate for the hidden layer state at instant t.
$x_t$ is the input to the hidden layer at instant t.
$h_t$ is the output state of the LSTM hidden layer at instant t.
$y_t$ is the final output at instant t, after applying the activation function σ to the hidden layer state $h_t$.
*tanh* function implements a non-linearity that squashes the activations to the range [-1,1].
$W_z$ and $b_z$ represents the combined connection weight and bias matrix between the previous hidden layer state $h_{t-1}$, present input $x_t$, and the update gate.
$W_r$ and $b_r$ represents the combined connection weight and bias matrix between the previous hidden layer state $h_{t-1}$, present input $x_t$, and the relevant gate.
$W_H$ and $b_H$ represents the combined connection weight and bias matrix between the product of previous hidden layer state $h_{t-1}$ with relevant gate state, present input $x_t$, and the candidate hidden layer state.
$W_Y$ and $b_Y$ represents the connection weight and bias matrix associated with the hidden layer $h_t$ and the activation layer.

### G. Bidirectional RNN Layer

If we want to have a mechanism in RNNs that offers comparable look-ahead ability, we need to modify the recurrent net design that we have seen so far. Fortunately, this is easy conceptually. Instead of running an RNN only in the forward mode starting from the first symbol, we start another one from the last symbol running from back to front. Bidirectional recurrent neural networks add a hidden layer that passes information in a backward direction to more flexibly process such information. Fig. 12 illustrates the architecture of a bidirectional recurrent neural network with a single hidden layer.

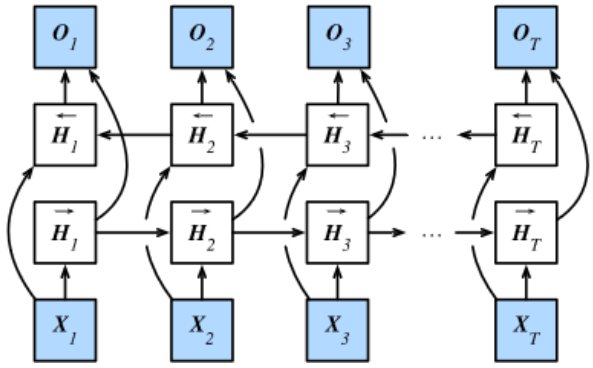


Fig. 12. Structure of a Bidirectional RNN hidden layer at time instant t. Here, $X_t$ represents the input to the bidirectional RNN at instant t. $H_t$ with → and ← symbol represents the hidden layers of forward and backward RNN respectively, which are concatenated to produce $O_t$ that represents the final output of the bidirectional RNN at instant t.

In fact, this is not too dissimilar to the forward and backward recursion we encountered above. The main distinction is that in the previous case these equations had a specific statistical meaning. Now they are devoid of such easily accessible interpretation and we can just treat them as generic functions. This transition epitomizes many of the principles guiding the design of modern deep networks: first, use the type of functional dependencies of classical statistical models, and then use the models in a generic form.

### H. Fully Connected Layer

Fully Connected layers are also known as dense layer. Here each hidden layer neuron is connected to every output neuron. They are usually connected with dropouts.

Dropout is a form of regularization. It aims to help prevent overfitting by increasing testing accuracy. For each mini-batch in the training set, dropout layers, with probability p, randomly disconnect inputs from the preceding layer to the next layer in the network architecture. Here, I randomly disconnect with probability p=0.4. After the forward and backward pass are computed for the minibatch, we re-connect the dropped connections, and then sample another set of connections to drop.

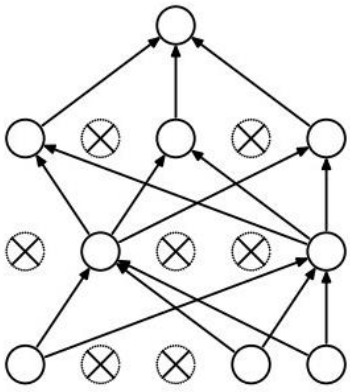

Fig. 13. Fully connected layer with dropouts.

### I. Activation Layer

A simple neural network just acts as a linear classifier. Therefore, to add nonlinearity to the neural network we use an activation function which decides whether a neuron should be activated or not by calculating the logit which is the weighted sum of the outputs of previous neuron and further adding bias with it. I have used the softmax activation function in my project.

The softmax function is an activation function that turns logits into probabilities that sum to one. I simple words, the output of all the neurons in a softmax layer represents a probability distribution. The activation value of $i^{th}$ neuron in the softmax layer is given by-

$$\sigma(z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, i \in 1,2,3,\dots,k \qquad (15)$$

where $z_i$ represents the logit from each neuron j in the layer.

### J. Training

For the training of each model I have considered the batch size to be 32 and number of epochs to be 10.

I have used the Adam optimizer to train the models. Adam is by far the most popular and widely used optimizer in deep learning. Adam is different to classical stochastic gradient descent which maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training. However, in Adam, a learning rate is maintained for each network weight (parameter) and separately

adapted as learning unfolds.

I have opted the binary cross entropy loss as a loss function to train the models. It is the loss function where the categorical outcome of the model is a binary variable. The algorithm will try to minimize this loss by adjusting model parameters/weights. The binary cross entropy loss will be equated as:

$$L=-\sum_{i}^{c} y_i \log \hat{y}_i \qquad (16)$$

where $c$ is the number of classes, $y_i$ is the actual for $i^{th}$ class, and $\hat{y}_i$ is the predicted value for the class.

## III. Results

### A. RNN Model

#### 1) Training

The accuracy vs epoch plot of the RNN model on the training and validation dataset is depicted in Fig. 14.

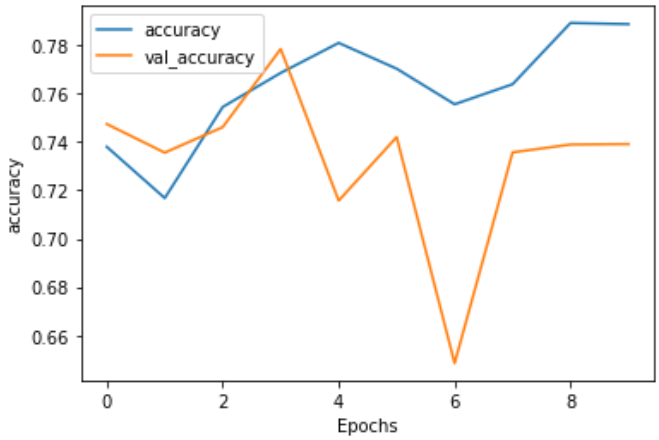


Fig. 14. Training and validation accuracy plot of RNN model.

The loss vs epoch plot of the RNN model on the training and validation dataset is depicted in Fig. 15.

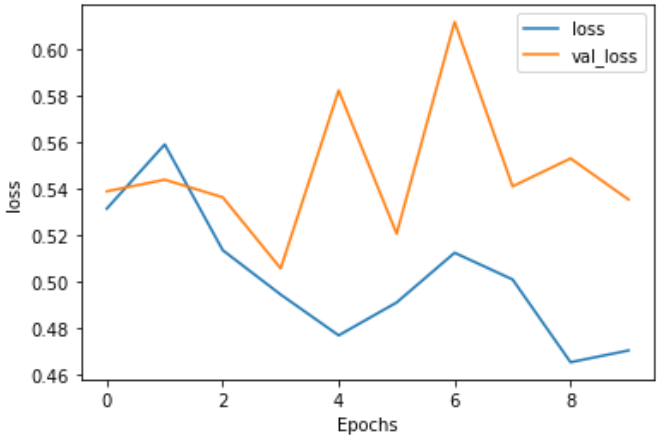


Fig. 15. Training and validation loss plot of RNN model.

#### 2) Testing

The RNN model achieved an accuracy of 77.85% with a loss score of 0.506 on the testing data. The confusion normalized matrix of the obtained RNN model is shown in Fig. 16.

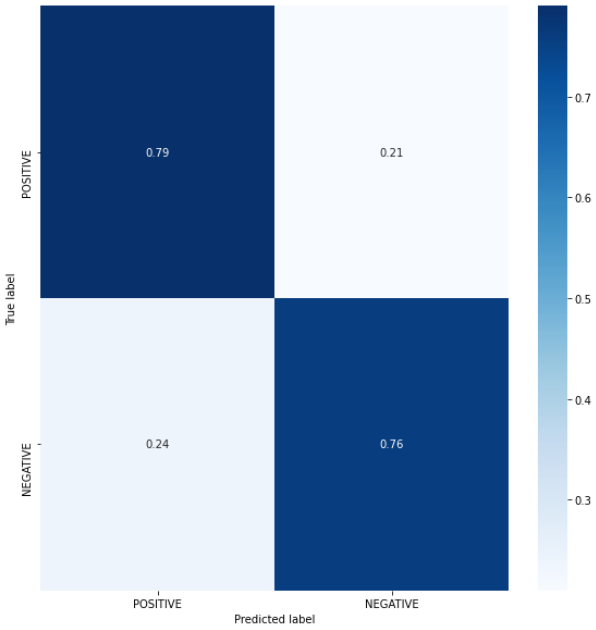


Fig. 16. Confusion matrix of the trained RNN model.

### B. LSTM Model

#### 1) Training

The accuracy vs epoch plot of the LSTM model on the training and validation dataset is depicted in Fig. 17.

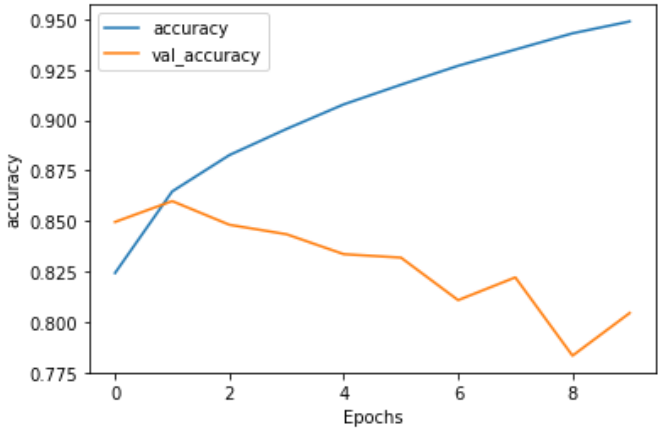


Fig. 17. Training and validation accuracy plot of LSTM model.

The loss vs epoch plot of the LSTM model on the training and validation dataset is depicted in Fig. 18.

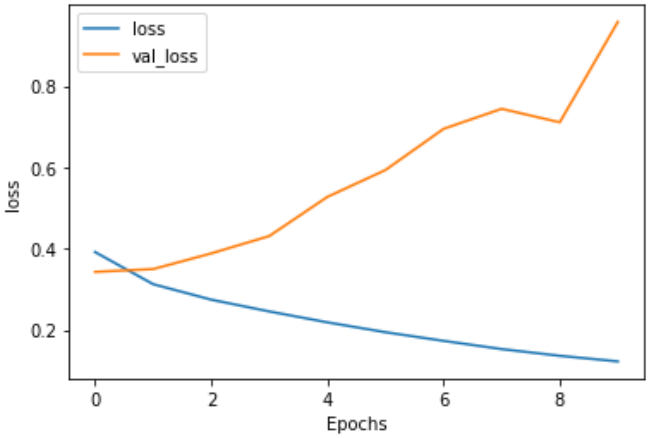

Fig. 18. Training and validation loss plot of LSTM model.

### 2) Testing

The LSTM model achieved an accuracy of 88.94% with a loss score of 0.323 on the testing data. The normalized confusion matrix of the obtained LSTM model is shown in Fig. 19.

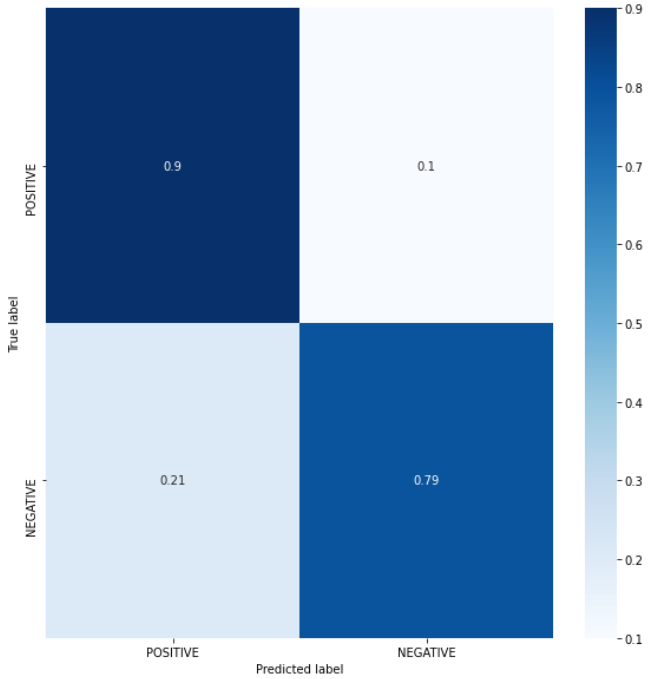


Fig. 19. Confusion matrix of the trained LSTM model.

## C. GRU Model

### 1) Training

The accuracy vs epoch plot of the GRU model on the training and validation dataset is depicted in Fig. 20.

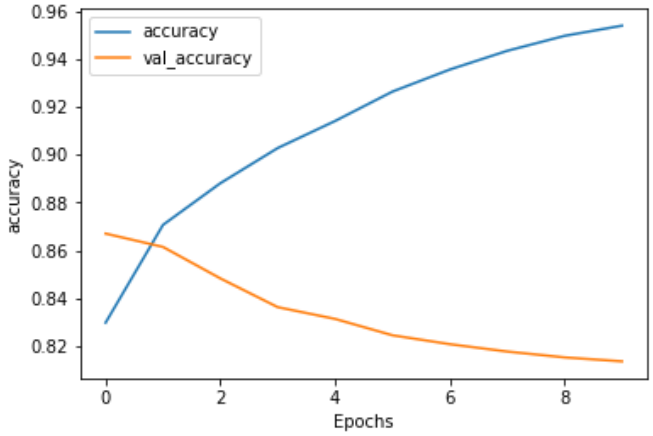


Fig. 20. Training and validation accuracy plot of GRU model.

The loss vs epoch plot of the GRU model on the training and validation dataset is depicted in Fig. 21.

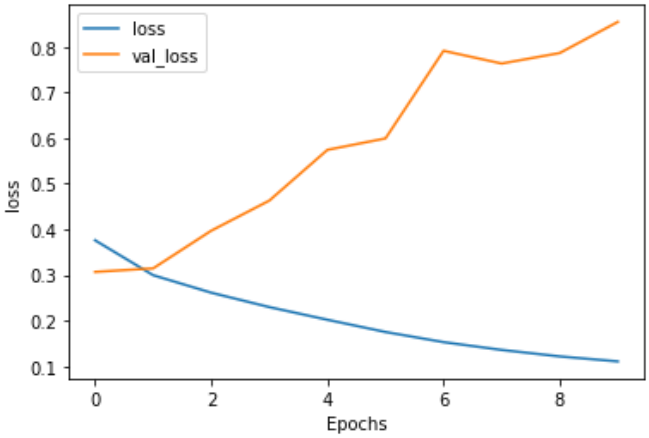


Fig. 21. Training and validation loss plot of GRU model.

### 2) Testing

The GRU model achieved an accuracy of 84.94% with a loss score of 0.342 on the testing data. The normalized confusion matrix of the obtained GRU model is shown in Fig. 22.

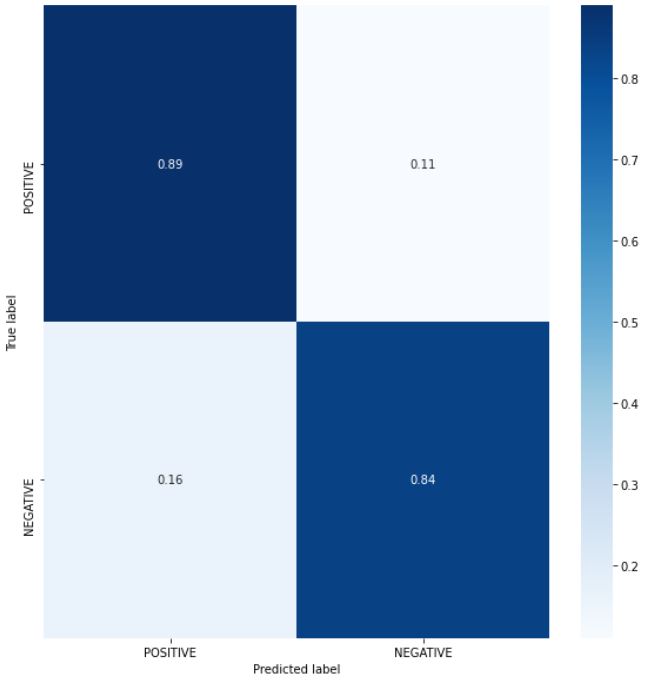


Fig. 22. Confusion matrix of the trained GRU model.

## D. Bidirectional RNN Model

### 1) Training

The accuracy vs epoch plot of the bidirectional RNN model on the training and validation dataset is depicted in Fig. 23.

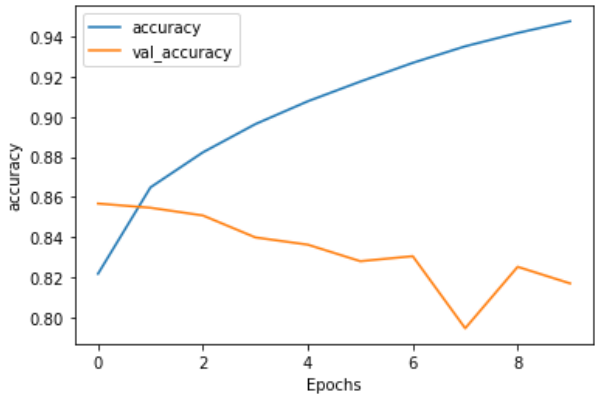


Fig. 23. Training and validation accuracy plot of bidirectional RNN model.

The loss vs epoch plot of the bidirectional RNN model on the training and validation dataset is depicted in Fig. 24.

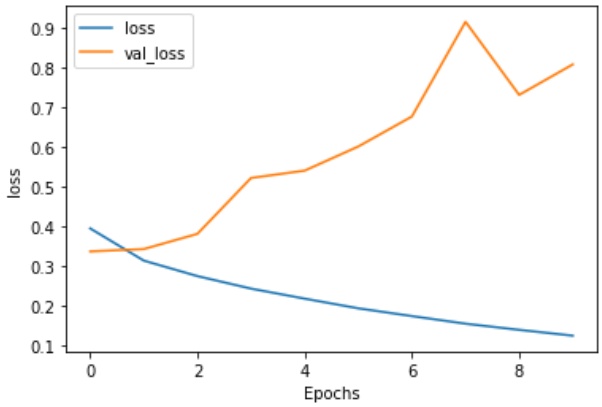


Fig. 24. Training and validation loss plot of bidirectional RNN model.

### 2) Testing

The bidirectional RNN model achieved an accuracy of 90.57% with a loss score of 0.317 on the testing data. The confusion normalized matrix of the obtained bidirectional RNN model is shown in Fig. 25.

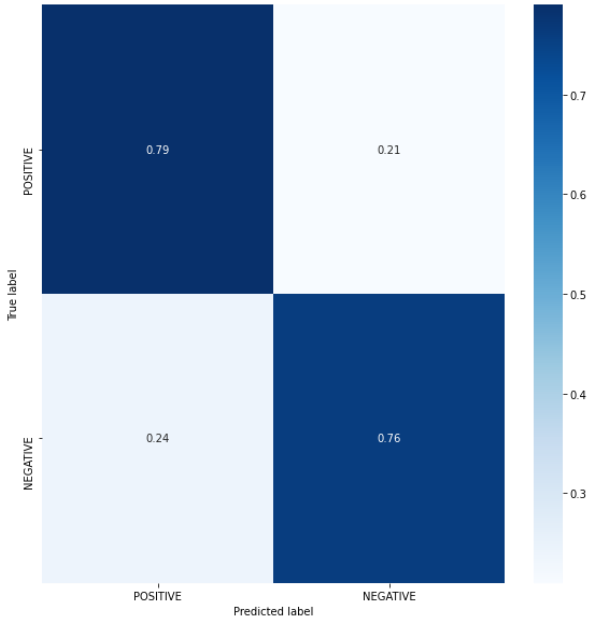


Fig. 25. Confusion matrix of the trained bidirectional RNN model.

## IV. Conclusion

RNNs are a great way to do sentiment analysis with minimum amount of workflow in natural language processing. However, for a larger sequence of input, RNNs may suffer from the problems like vanishing gradient and exploding gradient. It is clear from Fig. 14, the RNN model has a greater training accuracy than validation accuracy hence, the model is suffering from overfitting but here the difference is small as compared to other models.

LSTMs are more powerful than the RNNs due to their ability to efficiency in remembering long-term dependencies. This ability of LSTMs makes them overcome the vanishing gradient problem. The LSTM model have more test accuracy than the RNN model. It can be inferred from Fig. 17 that the LSTM model do suffer from overfitting but this time the difference between the training and validation accuracy is more as compared to the RNN model.

GRUs are simpler versions of LSTMs. The GRU model have a test accuracy similar to the LSTM model. As we can see from Fig. 20 that the GRU model is also subjected to overfitting similar to the LSTM model.

Bidirectional RNNs are more efficient than simple RNNs due to their look ahead ability. The bidirectional RNN model have a higher test accuracy than all of the models. But they are also subjected to a huge overfitting as shown in Fig. 23.

## Acknowledgment

I would like to thank Dr Muneendra Ojha, assistant professor, department of computer science, IIIT Naya Raipur, my research supervisor, to help me with his precious guidance and resources.

## References

[1] A. C. Ian Goodfellow, Yoshua Bengio, “Deep Learning Book,” *Deep Learn.*, 2015, doi: 10.1016/B978-0-12-391420-0.09987-X.

[2] C. Olah, “Understanding LSTM Networks [Blog],” *Web Page*, 2015, doi: 10.1007/s13398-014-0173-7.2.

[3] J. Chung, “Gated Recurrent Neural Networks on Sequence Modeling arXiv : 1412 . 3555v1 [ cs . NE ] 11 Dec 2014,” *Int. Conf. Mach. Learn.*, 2015.

[4] A. Graves and J. Schmidhuber, “Framewise phoneme classification with bidirectional LSTM and other neural network architectures,” in *Neural Networks*, 2005, doi: 10.1016/j.neunet.2005.06.042.

[5] A. Hassan, “SENTIMENT ANALYSIS WITH RECURRENT NEURAL NETWORK AND UNSUPERVISED Ph . D . Candidate : Abdalraouf Hassan , Advisor : Ausif Mahmood Dep of Computer Science and Engineering , University of Bridgeport , CT , 06604 , USA,” no. March, pp. 2–4, 2017.

[6] L. Arras, G. Montavon, K.-R. Müller, and W. Samek, “Explaining Recurrent Neural Network Predictions in Sentiment Analysis,” 2018, doi: 10.18653/v1/w17-5221.

[7] A. Shelar and C. Y. Huang, “Sentiment analysis of twitter data,” in *Proceedings - 2018 International Conference on Computational Science and Computational Intelligence, CSCI 2018*, 2018, doi: 10.1109/CSCI46756.2018.00252.

[8] D. Tang, F. Wei, N. Yang, M. Zhou, T. Liu, and B. Qin, “Learning Sentiment-Specific Word Embedding,” *Acl*, 2014, doi: 10.3115/1220575.1220648.