Rupal Bhargava*, Shivangi Arora and Yashvardhan Sharma

Neural Network-Based Architecture for Sentiment Analysis in Indian Languages

<https://doi.org/10.1515/jisys-2017-0398>

Received June 30, 2017; previously published online June 5, 2018.

Abstract: Sentiment analysis refers to determining the polarity of the opinions represented by text. The paper proposes an approach to determine the sentiments of tweets in one of the Indian languages (Hindi, Bengali, and Tamil). Thirty-nine sequential models have been created using three different neural network layers [recurrent neural networks (RNNs), long short-term memory (LSTM), convolutional neural network (CNN)] with optimum parameter settings (to avoid over-fitting and error accumulation). These sequential models have been investigated for each of the three languages. The proposed sequential models are experimented to identify how the hidden layers affect the overall performance of the approach. A comparison has also been performed with existing approaches to find out if neural networks have an added advantage over traditional machine learning techniques.

Keywords: Sentiment analysis, deep learning, Indian languages, convolutional neural network (CNN), recurrent neural network (RNN), long short-term memory (LSTM).

1 Introduction

Sentiments refer to the emotions of a person. There has been a surge in the number of people expressing themselves through tweets in different languages. Tweets have been used to give feedback in fields such as critical political issues, new movie releases, sports, etc. All of these fields need public feedback in order to work upon the glitches deduced from public reviews. It is necessary to know how many people are in favor (positive review) or against (negative review) the topic of discussion. Thus, analysis of tweets, in terms of the sentiments they reflect, is of great scope. According to Google in collaboration with KPMG (<https://assets.kpmg.com/content/dam/kpmg/in/pdf/2017/04/Indian-languages-Defining-Indias-Internet.pdf>), most of the people in India prefer tweeting in their regional languages, and hence, sentiment analysis of data in Indian languages has become significant. Sentiment analysis uses statistics, natural language processing (NLP) and machine learning techniques to predict the polarity of a sentence and gauge the correctness of the sentiment deduced.

There has been much research on English tweets but not on the tweets in Indian languages. This paper aims to predict the sentiments of monolingual tweets in one of the Indian languages (Hindi, Bengali, and, Tamil). The approach presented in the paper is a binary classification approach with the output being a label: 0 for negative sentiment and 1 for positive sentiment. Machine learning techniques extract features from a sentence to realize its polarity, but they do not consider the meaning attached to each word of a sentence [30]. To overcome this, word embeddings have been used to capture the semantic information of the text [34].

In this paper, neural networks have been used to identify the sentiment of a tweet. The major contribution of this paper is the use of all possible combinations up to three hidden layers of the three major neural network layers (recurrent neural network (RNN) [6], convolutional neural network (CNN) [14], and long

*Corresponding author: Rupal Bhargava, WiSoc Lab, Department of Computer Science, Birla Institute of Technology and Science, Pilani-333031, India, e-mail: rupal.bhargava@pilani.bits-pilani.ac.in

Shivangi Arora and Yashvardhan Sharma: WiSoc Lab, Department of Computer Science, Birla Institute of Technology and Science, Pilani-333031, India

short-term memory (LSTM) [11]) and analyzing the results in detail. For each of the three Indian languages, 39 hybrid models have been proposed, which are based on vector space. All the sequential models are tested using uniform parameters for the layers.

The rest of the paper has been organized as follows: Section 2 elaborates the related work. Section 3 explains the proposed approach and discusses the creation of different sequential models with optimum settings of parameters. Section 4 analyzes the sequential models proposed and discusses the results being obtained. It also compares the performance of the proposed approach to that of the approach based on traditional machine learning techniques. Section 5 concludes the paper.

2 Related Work

Sentiment analysis is a growing area of interest. Much research has been done to predict the polarity of sentences using classifiers like SVM, Naive Bayes, Decision trees, etc. These classifiers have been used in conjunction with word embeddings to produce state-of-the-art results. Most of the research has been focused upon data obtained from social media.

Giachanou et al. [7] presented a scrutinized report on most prevalent approaches for twitter sentiment analysis such as SVM and Naive Bayes along with data sets such as *Edinburg Twitter corpus*, *Stanford Twitter Sentiment*, *Sanders corpus*, etc., and sentiment lexicons such as SentiWordNet and MPQA. Da Silva et al. [4] presented hybrid models consisting of a combination of SVM, Naive Bayes, Decision Trees, etc., to achieve a state-of-the-art accuracy of 81.06%. A comparison between feature hashing and bag-of-words was also presented with a concluding remark that feature hashing seems to be a good choice when computational efficiency is the concern, while when accuracy is to be focused upon, bag-of-words seems to be the most appropriate choice.

Bootstrapping models were proposed by Hassan et al. [10] to bring consistency in results for imbalanced data obtained from Telco, Tch, and Pharma tweets. A two-stage process of expansion and contraction was used to produce models with better accuracy in predicting sentences with extreme polarity. The approach used by Araque et al. [1] combines traditional models that use manual feature extraction as surface models with deep learning techniques for sentiment analysis using word embeddings. Six public data sets were used for evaluation based on F-measure. A statistical analysis revealed that the ensemble performs better than the baseline model consisting of only deep learning-based classifier.

Muhammad et al. [21] worked upon not much explored Bangla text using the term frequency-inverse document frequency searching model to examine the polarity of a sentence as well as the sentiment of the whole document with respect to the subject. The data set used consists of around 1500 comments in Bangla from various social sites. Precision and recall were used to measure the correctness of the classifier in predicting the polarity of a sentence, while accuracy was used to judge the overall performance of the classifier.

Se et al. [27] used supervised machine learning techniques such as SVM, Maxent classifier, Decision Trees, and Naive Bayes to classify Tamil movie reviews as positive or negative. It was observed that SVM outperformed all other models in terms of accuracy (which was calculated to be 75.9%) and cross-validation followed by Decision Trees. The labels used were: -1 for negative polarity and +1 for positive polarity. The experiment was performed with and without the use of TamilSentiWordNet. The highest accuracy obtained using TamilSentiWordNet was 4% more than that obtained without it.

The approach proposed by Uma et al. [31] used the SVM classifier to retrieve information from Tamil and English tweets and classify each sentence as positive, negative, or neutral. WordNet was used to capture the meaning of a word, and the semantic similarity between words was calculated by measuring the path-length-based similarity. Twitter data collected using TwitterAPI was used as data set. Confusion matrix was obtained for evaluation and, *weka* was used to perform classification of tweets using SVM. StringToWord-Vector was used in the pre-processing step. The F-measure obtained using SVM was 0.741, while it was 0.68 using Naive Bayes, thereby, signifying SVM is a better choice over Naive Bayes for sentiment classification of textual data.

Phani et al. [23] performed sentiment classification on the Sentiment Analysis of Indian Languages (SAIL) 2015 data. Two types of classification experiments were adopted: (1) two-class: positive and negative classes; (2) three-class: positive, negative, and neutral classes. Stratified 10-fold cross-validation was used on training data. The models performed extremely well compared to models in SAIL 2015 with an accuracy hitting 56.96% for Hindi, 51.25% for Bengali, and 45.24% for Tamil.

3 Proposed Approach

The proposed approach is a word-level binary classification approach, which uses vector-space model to predict the sentiment of a tweet. It is divided into three phases: pre-processing, creation of sequential model, and predictions. These phases convert the textual data to a label, which determines the sentiment of a tweet.

3.1 Phase I: Pre-Processing

Given a set of L sentences, $S = \{S_1, S_2, \dots, S_L\}$, each sentence S_i , with n words $\{w_1, w_2, \dots, w_n\}$, $n \in \mathbb{R}$, where $\mathbb{R}$ denotes a set of real numbers, is broken down into its constituent words, and each word is tokenized, thus, giving a dictionary of words $w_t = \{w_1, w_2, \dots, w_n\}$. Each word or token is then assigned an index on frequency basis of its occurrence in the text file. The indices are then concatenated to return a numerical array for each sentence. The sentences in the form of numerical arrays are then padded to yield sentences of similar dimensions, filling the extra columns added to equalize the dimensions with 0, and returning a padded array, w_p .

3.2 Phase II: Sequential Model

The padded numerical array, w_p , of each sentence, is used as input to the Sequential model, a linear stack of layers. Each index in the array w_p is mapped to real-valued vectors, $v_i \in \mathbb{R}^d$ where v_i is a d -dimensional vector in high-dimensional space using the embedding layer. This produces a matrix of feature vectors (word embeddings), V , that captures the semantic and syntactic information about the words. The matrix of vectors is represented in equation (1):

$$V = [v_1, v_2, \dots, v_{n+k}]. \quad (1)$$

$$= \begin{bmatrix} v_1^1 & v_1^2 & \dots & v_1^{n+k} \\ v_2^1 & v_2^2 & \dots & v_2^{n+k} \\ \vdots & \vdots & \vdots & \vdots \\ v_L^1 & v_L^2 & \dots & v_L^{n+k} \end{bmatrix}. \quad (2)$$

where, each row represents vectors of a sentence.

The output of the embedding layer, the word embeddings, is input to further layers in the model. The hybrid of three neural network layers (RNN, LSTM, CNN) is used to create 39 different models.

3.2.1 Convolutional Neural Network

CNN [14] uses a filter $F \in \mathbb{R}^n$ over a window of n words to extract overlapping features throughout the training and testing set consisting of vectors, v_i . The convolution of F and v_i produces a new feature, c_i :

$$c_i = f(\mathbf{F} \cdot v_i) \quad (3)$$

where, f is a non-linear function and ' $\cdot$ ' is element-wise multiplication.

The new features, $c = \{c_1; c_2; \dots; c_i\}$ form a feature map. This feature map is passed as an input to the max-pooling layer. This layer uses non-overlapping filter of length ' a ' (in the case of one-dimensional pooling layer) and reduces the size of the feature matrix by keeping only the highest values in each chunk [33].

3.2.2 Recurrent Neural Network

RNN [6] uses a memory feature to capture and remember the information it has learnt so far. It takes the i^{th} word as input and predicts the probability of $(i+1)^{\text{th}}$ word [28]. This is done using a non-linear function. RNN produces a new embedding matrix, including the information from the previous embedding [12].

3.2.3 Long Short-Term Memory

LSTM [11] is devised to overcome the problem of vanishing and exploding gradients in RNN [6]. It uses memory cells with three gates: input, output, and forget gate. It chooses which information to keep or forget by assigning appropriate weights to the gates. It also produces a new embedding matrix using the information from the previous one.

As can be seen, each neural network works in a slightly different way from the other. When using a hybrid of these neural networks, one can increase the efficiency of the model by combining the advantages of networks. For example, using CNN along with LSTM allows CNN to learn the features using overlapping sequences resulting in a matrix of word vectors that contributes majorly to the sentiment of the sentence. This matrix is fed to LSTM, which can focus on more important data rather than memorizing the whole sentence. This increases the efficiency of the model. In some cases, combining neural networks can help form bi-directional networks, which in turn helps predict future words that play a vital role when dealing with data of high complexity. The stacking of layers can increase the non-linearity of the model to produce better decision boundaries and, hence, more reliable results.

Optimum parameters have been selected for each layer, which remain fixed across all the models in order to compare the results at the end of the experiment. Table 1 shows the parameter settings for different layers.

In sequential model, two dense layers have been appended after the hybrid layers with different neuron units as shown in Table 1. These layers are fully connected layers of the model containing activation functions that connect each input neuron to each neuron of the next layer. A flattened layer is used before the dense layers in order to convert the dimensions of the output from previous layers, to usable form in dense layer. To avoid over-fitting, two dropout layers have been applied to each of the dense layer to drop some of the connections of neurons from the dense layer (as shown in Figure 1). The activation functions used for dense layers are *ReLU* and *Sigmoid* as they do not suffer from saturation. The *Sigmoid* function as shown in equation (4),

$$S = 1 / (1 + e^{-x}) \quad (4)$$

produces an output from 0 to 1. It converts the embedding matrix from RNN and LSTM or output from max-pooling layer to a label 0 or 1, which represents negative and positive sentiments, respectively. The model is finally compiled with a loss function to yield a binary classification.

The whole process of sentiment analysis from a set of sentences to generating a label is shown in the steps in algorithm 1. In algorithm 1, $\mathbf{Y}$ represents a matrix of labels for each sentence in test data ($\mathbf{Y}_{\text{ts}}$) and training data ($\mathbf{Y}_{\text{tr}}$).

$$Y = [y_1, y_2, \dots, y_L] \quad (5)$$

where, each column n of the matrix represents label for a given sentence.

Table 1: Parameter Settings for Sentiment Analysis.

Sr. no	Layers	Parameters
1	Embedding	input_dim = 5000, output_dim = 20, input_length = 30
2	Simple RNN	Neuron units = 100
3	LSTM	Neuron units = 100
4	CNN	nb_filter = 20, filter_length = 3, activation = 'relu'
5	Max-pooling	pool_length = 2
6	Dropout(Layer 1)	units: Hindi: 0.8 Bengali: 0.7 Tamil: 0.7
7	Dropout(Layer 2)	units: Hindi: 0.5 Bengali: 0.3 Tamil: 0.3
8	Dense(Layer 1)	units = 20, activation = 'relu'
9	Dense(Layer 2)	units = 1, activation = 'sigmoid'
10	model.compile()	Loss = 'binary_crossentropy', optimizer = 'adam'
11	model.fit()	batch_size = 32, epochs = 8
12	model.predict()	batch_size = 32, verbose = 1
13	model.evaluate()	verbose = 1

Figure 1: Block Diagram Representing the Process of Sentiment Analysis.**Algorithm 1:** Sentiment Analysis of Indian Languages.

Input: $x = A$ set of sentences, $S = S_1, S_2, \dots, S_n$ with n words, $\{w_1, w_2, \dots, w_n\}$, of one of the Indian languages
Output: $y \in \{\text{positive: 1 negative: 0}\}$

- 1 **Assumption:** Each document is a monolingual data set
- 2 **for each** S_i **in** S **do**
- 3 **Phase I:** Pre-Processing
- 4 1. Tokenize
- 5 $w_t = f(S_i)$, $S_i = \{w_1, w_2, \dots, w_n\} \in S$
- 6 Output: $w_t = \{'w_1', 'w_2', \dots, 'w_n'\}$
- 7 2. Pad
- 8 $w_p = g(w_t)$, $w_t = \{'w_1', 'w_2', \dots, 'w_n'\}$
- 9 Output: If words in the largest sentence = $n + k$
- 10 $w_p = \{'w_1', 'w_2', 'w_n', 0, 0, 0 \dots k \text{ times}\}$
- 11 **Phase II:** Sequential Model
- 12 **If** S_i **belongs to training data then**
- 13 Feed the data: model.fit(Y, Y_{tr})
- 14 **else**
- 15 Feed the data: model.predict(Y, Y_{ts})
- 16 **end**

$$y_i = \begin{cases} 0, & \text{for negative } S_i \\ 1, & \text{for positive } S_i \end{cases} \quad (6)$$

3.3 Phase III: Predictions

This phase operates only on the test data. The test data is fed to the sequential model using `model.predict()` as explained in algorithm 1. The sequential model works upon it and produces labels for each sentence in the test data set.

4 Experiment and Results

4.1 Sentiment Analysis Data Sets

The data set used has been extracted from the SAIL 2015 data [22], which contains twitter data from three Indian languages: Hindi, Bengali, and Tamil. Each data set is divided into two text files: a positive sentence data file and a negative sentence data file. The whole data set is split into training and testing data in the ratio of 70:30. Table 2 enlists the number of sentences in the training and testing set for each of the three Indian languages. The data set contains monolingual tweets in one of the Indian languages. The textual data is subjected to some transformations referred to as the pre-processing phase, which is discussed in Section 3.

4.2 The Experimental Setup

The experiment aims at predicting the sentiment of sentences in one of the three Indian languages mentioned before and gauges the correctness of the model using performance measures. For this purpose, all possible combinations of RNN, LSTM, and CNN have been modeled, and performance measures such as accuracy, F-measure, precision, and recall have been calculated. Over-fitting has been visualized using the accuracy v/s number of epochs graph.

The experiment uses a vector-space-based model that encodes continuous similarities between words as distances or angle between word vectors in a high-dimensional space [18]. Analysis of short texts is difficult due to limited amount of contextual data available in the text. Thus, to fill the gap of contextual information in a scalable manner, it is more suitable to use methods that can exploit prior knowledge from large sets of unlabeled texts [5]. Hence, word embeddings have been used as the first layer, i.e. the word embedding layer. It takes as input an array of padded numerical arrays of sentences with each sentence padded to 30 integral words and transforms each one of them to an array of feature vectors, each vector being 20-dimensional. The embedding layer has been trained using a part of the data set held as training data. The integral words are

Table 2: Data Set Description in Terms of Number of Sentences.

Sr. no	Data set	Class	Total data	Training data	Test data
1	Hindi	Positive	168	508	219
		Negative	559		
2	Bengali	Positive	277	441	190
		Negative	354		
3	Tamil	Positive	387	492	211
		Negative	316		

projected as random vectors in the 20-dimensional space. The model is kept trainable so that these vectors get updated and optimized in the word-vector space during back-propagation while capturing the context of each word.

The next layer is a neural network, which can be a single layer or a combination of RNN [6], LSTM [11], and CNN [14]. This network takes feature vectors as input and processes it. For example, convolutional layer takes feature vectors as input, breaks them into overlapping sequences, which are read using a filter of fixed window, and a feature map is prepared. This feature map is fed to max-pooling layer, which returns the largest number in the array as output. This output is fed to dense layers where activation functions are applied on them, and a final output is given, which is a label (0: negative; 1: positive) [33]. Figure 1 shows the block diagram of the process occurring in the experiment.

4.3 Parameter Setting

To train the model, the number of epochs have been set to eight because below this value, either the model is over-fitting or the performance measures are exactly similar. Over-fitting occurs when the training error is decreasing at a higher rate than the testing error, and thus, the gap between training and testing accuracies is increasing. This is the case when epochs are greater than eight. The gap between training and testing accuracies remains the same at eight epochs in the same cases, whereas it is decreasing in some others. Only a few models (three or four out of 39, that, too, mostly three-layered networks) underwent over-fitting. However, for the Tamil data set, there is no over-fitting at eight and seven epochs, but the results for seven epochs are better than that at eight, and hence, seven epochs have been selected for the Tamil data set. A batch_size of 32 has been selected as the data set contains short texts of single line.

4.4 Results and Evaluation

For evaluating the performance of sentiment analysis, accuracy and F-measure are mainly used to compare the results of different models [29]. Accuracy helps to know how accurate the predicted results are, which is important when using sentiment analysis for business interests, political issues, etc. The F-measure gauges how well the system is performing, which is again needed to keep a track of the system's consistency, and also, it takes into account precision and recall. The results for the different data sets are discussed below:

4.4.1 Hindi Data Set

The results of the top performing models are summarized in Table 3. Figure 2A–C shows the accuracy and F-measure of all the models. The single, dual, or triple layer models of the CNN performed well in terms of accuracy due to multiple reasons. First, the CNN has a non-linear activation function (*ReLu*) in itself, while any other model does not. This captures element-wise non-linearity. Second, the CNN derives n -grams (sequence

Table 3: Summary of Top Performance Results for Hindi Data Set.

Sr. no	Model	Accuracy	F-measure
1	CNN	77.63%	67.85%
2	RNN–LSTM	77.63%	67.85%
3	LSTM	74.89%	69.08%
4	CNN–LSTM–LSTM	76.26%	70.50%
5	LSTM–RNN–LSTM	74.43%	68.8%

Figure 2: Accuracy and F-measure for Hindi Data Set.

(A) One and two layer models, (B) three layer models (part-1) and (C) three layer models (part-2).

of n words) and forms feature maps using them, which are then fed as the input to the max-pooling layer. These n -grams provide non-linear interactions within a local context and improve the performance of the model [17]. Third, the CNN uses multiple filters with different window sizes that move over the word embeddings to perform one-dimensional convolution. As the filter moves on, many sequences, which capture the semantic and syntactic features in the filtered n -grams, are generated. Many such features are combined into the feature map, and further operations are performed in the pooling layer, which only increases the efficiency and accuracy of the results drastically. This allows the model to capture the sentiment of the phrases such as ‘not so good’. Fourth, the network can easily explore the richness of the word embeddings produced by unsupervised pre-training [5]. Fifth, for small training data, the CNN performs better than any other model [13]. The models, though, lagged behind in terms of F-measure, which indicates that there is no consistency of the system during the iterations (as shown in Figure 2A and C). When precision and recall are looked upon separately, it is found that these models lacked precision but have high recall scores. This signifies a high false-positive value indicating incorrectly predicting sentiments as positive for some cases, which can probably be due to skipping of the low-level features by the CNN, which could have played a major part in predicting the sentiment. This is also the reason why the CNN, along with the LSTM or RNN, performed better as the RNN or LSTM is able to process low-level features as well.

The RNN-LSTM model performed equally well as the CNN model. It acts as a bidirectional RNN model, which has been shown to improve the performance of a model [25]. The feature vectors from the embedding layer are fed to the RNN, which reads the whole sentence and stores in its memory, though it is short term. The output from the RNN will be fed to the LSTM, which has memory cells that control whether to keep the data, read it, or forget it. It does so by assigning some weights to the data. However, the problem may occur when this results in loss of information due to error persistence in the LSTM as the LSTM’s memory cells are linear neurons and cause the error to be carried forward for a longer time. This problem can be overcome using the RNN layer before the LSTM. As the RNN has nodes inter-connected to other nodes in conjunction

to themselves, the information lost from the LSTM can be stored in the RNN for a while so that it is not lost and can be propagated further if required during discrepancies [9]. Thus, this network takes care of loss of information (using RNN) and loss of efficiency (using LSTM, which can adapt to new data rapidly [8]). The LSTM and LSTM–LSTM performed really well due to the ability of the LSTM to adapt to new data rapidly [19], but the triple-layered LSTM underwent some over-fitting (as shown in Figure 2A and B).

The RNN–CNN–LSTM performed pretty well with an accuracy of 74.43% and an F-measure of 68.23% as shown in Figure 2B. The CNN and the LSTM are complimentary, where the CNN captures the local invariant regularities, the LSTM is good at modeling temporal features. The RNN helps to maintain the memory of the words in the sentence [32]. The RNN–RNN–CNN gave an accuracy of only 53.88% (as shown in Figure 2B), which is the least among all the models and is also exceptionally low as all other accuracies range from 60% to 70%. This is because the model underwent over-fitting with the testing accuracy decreasing at a very high rate and the training accuracy increasing as shown in Figure 5A. The RNN and RNN–RNN did not perform well compared to the CNN and CNN–CNN as shown in Figure 2A. The LSTM follows the same trend. This is probably because in the case of the RNN and LSTM, increasing the number of layers would only increase the complexity and error. This is due to linear stacking of the errors that reduce the accuracy as the number of layers increase. While in the case of the CNN, increasing the number of layers increases the ability of the CNN to process even the smallest but relevant information, which can otherwise be missed during convolution. This means that increasing the number of layers of the CNN should increase the accuracy, but as the number of layers increase, the error in classification also increases [32], thus, compensating for the increase in accuracy and giving the same accuracy.

4.4.2 Bengali Data Set

The Bengali data set is small compared to the Hindi data set, and hence, to avoid over-fitting, the dropout units are reduced as shown in Table 1. The number of epochs have been kept the same as discussed in Section 3.

Figure 3: Accuracy and F-measure for Bengali Data Set.

(A) One and two layer models, (B) three layer models (part-1) and (C) three layer models (part-2).

Table 4: Summary of Top Performance Results for Bengali Data Set.

Sr. no	Model	Accuracy	F-measure
1	LSTM–CNN	57.37%	45.8%
2	RNN–LSTM	56.84%	45.13%
3	CNN–RNN	56.32%	55.60%
4	RNN–CNN–CNN	57.37%	57.1%
5	RNN–LSTM–LSTM	54.21%	53.80%

According to the results shown in Figure 3A to C, models with two hidden layers are more successful compared to single or triple hidden layer models. This is because Bengali is a bit more complicated than Hindi [2], and therefore, the feature extraction of more curvi-linear fonts in Bengali need more complex models. Hence, the dual-layered models performed better than the single-layered models, but the triple-layered models underwent error accumulation and, in some cases, over-fitting due to a small data set. Table 4 summarizes the results for top performing models. The CNN models did not perform well when compared to the CNN models in the Hindi data set. The LSTM–CNN network is an excellent network for query classification as the CNN and LSTM are complimentary, where the CNN captures the local invariant irregularities, the LSTM equipped with input, output, and forget gates, extract the first-order feature representation using the memory of the whole sentence. This is essential for a complicated language like Bengali (This feature is missing in the RNN, and hence, the performance of the RNN–CNN is not at par with that of the LSTM–CNN.). The LSTM can adapt to new sentences quickly, which gives it an upper edge over models utilizing only the CNN layers for sentiment classification of a new sentences [8]. This is utilized along with great feature extraction by the CNN to give a higher accuracy.

The RNN–CNN–CNN has an accuracy of 57.37% and an F-measure of 57.1% (as shown in Figure 3C), which seems to be the highest among all the models, but it turns out that the model is over-fitting due to a small data set as shown in Figure 5B. The training accuracy is increasing at a greater rate than the test accuracy, which is a case of over-fitting.

In the Bengali data set as well, the CNN models did not perform well in terms of F-measure for the same reason as explained in the Hindi data set. The RNN–RNN–CNN, too, has a very low F-measure due to less consistency of the results, which means it predicted some sentiments as positive while they are actually negative. This could have been due to small training data.

4.4.3 Tamil Data Set

The Tamil data set is small compared to the Hindi data set, and hence, dropout units are reduced in order to avoid over-fitting as shown in Table 1. The results can be seen in Figure 4A–C. Table 5 summarizes the results for the top performing models. As can be seen from Table 5, three hidden layer models gave the best results without over-fitting. This may be due to the fact that Tamil is even more complicated than Bengali and Hindi [20] due to its curvi-linear and multi-tier structure with a contrast of *matra* combinations. Hence, it needs more complex models to get its features extracted to analyze the sentiment of its sentences.

The RNN–CNN–LSTM outperformed all the other models according to the results, but it turns out that the model is over-fitting with the training accuracy increasing at a greater rate than the testing accuracy as shown in Figure 5D. Hence, the top model of the proposed approach becomes RNN–LSTM–RNN, which did not over-fit and gave the highest accuracy as well as F-measure.

The RNN–LSTM performed well on the Tamil data set as well, as shown in Figure 4A. The details of this model have been discussed in Section 4.4.1. Adding a layer of the RNN to this model only increases the complexity, which is needed in order to extract the features from Tamil sentences. The RNN would not filter the words unlike the CNN and LSTM but only provide memory to the model. It acts as a bidirectional RNN–LSTM,

Figure 4: Accuracy and F-measure for Tamil Data Set.

(A) One and two layer models, (B) three layer models (part-1) and (C) three layer models (part-2).

Table 5: Summary of Top Performance Results for Tamil Data Set.

Sr. no	Model	Accuracy	F-measure
1	RNN-CNN-LSTM	71.56%	69.7%
2	RNN-LSTM-RNN	69.19%	68.7%
3	RNN-RNN-LSTM	68.72%	68%
4	RNN-LSTM	67.30%	66%
5	CNN-LSTM-CNN	67.77%	49%

which is important in cases when future information is as important as the previous information [25]. The same reasoning is applicable to the RNN-RNN-LSTM model whose accuracy is almost equal to the RNN-LSTM-RNN, and the F-measure is exactly the same (as shown in Figure 4B). Using the LSTM as the common layer increases the accuracy in the case of the RNN-LSTM-RNN due to its tendency to keep a memory of long sentences and filtering techniques.

The CNN-LSTM-CNN performed well in terms of accuracy but lagged a great deal in terms of F-measure and, hence, cannot be considered as one of the best models as the results are being analyzed considering both accuracy and F-measure. The RNN-RNN went over-fitting (as shown in Figure 5C) and produced an accuracy score, which is nowhere close to any other model. It attained its highest accuracy in the early epochs and went over-fitting thereafter. The RNN-RNN-CNN gave a very low F-measure due to reasons presented in Section 4.4.1 on the evaluation of the Hindi dataset.

It is observed that on all the three data sets, the RNN-LSTM performed well, but the accuracies and F-measure are different as shown in Figure 6. One can deduce from the bar graph that the more data are available in a data set, the better is the performance of a given model on the data set. It also shows that Hindi being the least complex language [2], to extract features from, performance measures for the Hindi data set are better than for the other two languages. It is worth noting that although Tamil is more complicated than Bengali in terms of language complexity [20], the Tamil data set showed more accurate results than the Bengali data set. This is because the data available in the Tamil dataset are more than those in the Bengali data set, and hence, the more data are available, the more accurate are the results.

Figure 5: Over-Fitting in Models.

(A) RNN-RNN-CNN (Hindi), (B) RNN-CNN-CNN (Bengali), (C) RNN-RNN (Tamil) and (D) RNN-CNN-LSTM (Tamil).

Figure 6: Comparison of Performance of RNN-LSTM on Different Data Sets.

4.5 Comparison with Baseline Approaches

The approach of the best results obtained in the SAIL 2015 shared task [16] has been used as the baseline approach. In the SAIL, most of the participants used SentiWordNet (ILs) for training [23], which has not been used in the approach adopted in this paper. The authors extracted word unigrams and bigrams from the data set, trigrams and quad-grams for word prefixes and suffixes. The SentiWordNet features were

Table 6: Comparison with Kumar et al. [16].

Data set	Proposed approach	Kumar et al. [16]
Hindi	68.51%	51.58%
Bengali	52.62%	41.54%
Tamil	59.88%	28.20%

also used. To take into account the rare and unseen words, lexical expansion was used [16]. However, the approach proposed in this paper gives better results (as shown in Table 6) than any of the models used in the SAIL 2015 [24, 26]. This urges to find out what makes a proposed approach perform better. A straightaway difference is the use of neural networks in this experiment instead of traditional machine learning techniques used in the SAIL 2015. The highest accuracy scored [16] is using a fivefold cross-validation technique using SVM. In this paper, the K-fold cross-validation has not been used but a simple train/test split method due to the following reasons [15]:

1. The K-fold cross-validation is computationally expensive and, so, is not worth the trouble for validating 39 different models for each of the three languages.
2. The data sets used in this paper are small. The K-fold cross-validation generally undergo under-fitting on small data sets.

The approach used in Kumar et al. [16] has been implemented as mentioned in the paper. The accuracy for baseline approach [16] has been calculated using the data set mentioned in Section 4.1. The results are remarkably good for hybrid models (as shown in Table 6) compared to the highest accuracy model [16]. The advantages of neural networks over traditional machine learning techniques are as follows:

First, to predict the sentiment of a sentence, there is a need to look upon the sentence as a whole. The sentiment of a sentence can be quite different from the sentiment of its constituent words. Therefore, a memory factor is required to remember the previous and the future (in the case of bidirectional neural networks) words to predict the probability of the present word. This can be achieved using neural networks but not using any other model.

Second, traditional machine learning techniques are non-parametric models and cannot learn highly complex functions. On the other hand, neural network is a deep architecture and is able to combine lower-level features to high-dimensional representations. Using the concept of co-occurrence dependencies, i.e. words having similar sentiments occur together in high-dimensional space, neural network is able to make progress on high complexity tasks without human intervention [3].

5 Conclusion and Future Work

The approach in this paper presents the sentiment analysis of the SAIL 2015 data consisting of monolingual tweets in one of the three Indian languages (Hindi, Bengali, and Tamil). Word embeddings have been used as input to the sequential model. The approach investigates 39 different hybrid models of neural network layers (RNN, CNN, and LSTM). The highest accuracy has been obtained using the CNN (77.63%) and RNN–LSTM (77.63%) on the Hindi dataset, the LSTM–CNN (57.37%) on the Bengali dataset, and the RNN–LSTM–RNN (69.19%) on the Tamil dataset. The results have been closely scrutinized to conclude that as the complexity of the text in the data set increases, the accuracy decreases for each model. Models with more number of hidden layers provide better accuracy for more complex text as an increase in the number of hidden layers adds to the non-linearity of the model to predict complex situations. It has also been observed that neural networks perform better than other shallow architectures such as SVM, Decision Trees, etc., in predicting the polarity of a sentence.

Presently, pre-processing of the textual data is done to remove the emoticons, exclamations. However, they may be playing a role in determining the sentiment of the sentence. For example: ‘What!’ and ‘What?’ represent two different meanings, which cannot be distinguished using present-day sentiment analysis approaches as they tend to pre-process the sentences to remove the punctuation. Some people use only emoticons to express their sentiments, which can be embedded and worked upon for making the sentiment analysis approaches even more reliable. This work can also be extended to classify sentences into multiple classes such as extremely negative, negative, neutral, positive, and extremely positive, which gives a more precise sentiment to a sentence.

Bibliography

- [1] O. Araque, I. Corcuera-Platas, J. F. Sánchez-Rada and C. A. Iglesias, Enhancing deep learning sentiment analysis with ensemble techniques in social applications, *Expert Syst. Appl.* **77** (2017), 236–246.
- [2] S. Bag and G. Harit, Topographic feature extraction for Bengali and Hindi character images, *arXiv preprint arXiv:1107.2723* (2011).
- [3] Y. Bengio, Y. LeCun, Scaling learning algorithms towards AI, in: *Large-Scale Kernel Machines*, L. Bottou, O. Chapelle, D. DeCoste, J. Weston, eds., MIT Press, Cambridge, MA, USA, 2007.
- [4] N. F. F. Da Silva, E. R. Hruschka and E. R. Hruschka, Tweet sentiment analysis with classifier ensembles, *Decis. Support Syst.* **66** (2014), 170–179.
- [5] C. dos Santos and M. Gatti, Deep convolutional neural networks for sentiment analysis of short texts, in: *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, Dublin, Ireland, pp. 69–78, 2014.
- [6] J. L. Elman, Finding structure in time, *Cogn. Sci.* **14** (1990), 179–211.
- [7] A. Giachanou and F. Crestani, Like it or not: a survey of twitter sentiment analysis methods, *ACM Comput. Surv.* **49** (2016), 28.
- [8] A. Graves, Generating sequences with recurrent neural networks, *arXiv preprint arXiv:1308.0850* (2013).
- [9] D. Hakkani-Tür, G. Tür, A. Celikyilmaz, Y.-N. Chen, J. Gao, L. Deng and Y.-Y. Wang, Multi-domain joint semantic frame parsing using bi-directional RNN-LSTM, in: *INTERSPEECH*, pp. 715–719, 2016.
- [10] A. Hassan, A. Abbasi and D. Zeng, Twitter sentiment analysis: a bootstrap ensemble framework, in: *Social Computing (SocialCom), 2013 International Conference on*, IEEE, pp. 357–364, 2013.
- [11] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Comput.* **9** (1997), 1735–1780.
- [12] H. Jaeger, *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach*, 5, GMD-Forschungszentrum Informationstechnik Bonn, 2002.
- [13] R. Kadlec, M. Schmid and J. Kleindienst, Improved deep learning baselines for ubuntu corpus dialogs, *arXiv preprint arXiv:1510.03753* (2015).
- [14] Y. Kim, Convolutional neural networks for sentence classification, *arXiv preprint arXiv:1408.5882* (2014).
- [15] T. Krueger, D. Panknin and M. Braun, Fast cross-validation via sequential testing, *J. Mach. Learn. Res.* **16** (2015), 1103–1155.
- [16] A. Kumar, S. Kohail, A. Ekbal and C. Biemann, IIT-TUDA: system for sentiment analysis in Indian languages using lexical acquisition, in: *International Conference on Mining Intelligence and Knowledge Exploration*, Hyderabad, India, Springer, pp. 684–693, 2015.
- [17] T. Lei, R. Barzilay and T. Jaakkola, Molding CNNs for text: non-linear, non-consecutive convolutions, *arXiv preprint arXiv:1508.04112* (2015).
- [18] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng and C. Potts, Learning word vectors for sentiment analysis, in: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, Association for Computational Linguistics, pp. 142–150, 2011.
- [19] T. Mikolov, K. Chen, G. Corrado and J. Dean, Efficient estimation of word representations in vector space, *arXiv preprint arXiv:1301.3781* (2013).
- [20] S. K. Mohanty, *The formulation of parameters for type design of Indian scripts based on calligraphic studies*, Electronic Publishing, Artistic Imaging, and Digital Typography, St. Malo, France, Springer, pp. 157–166, 1998.
- [21] M. N. Muhammad, T. Altaf Md. and S. Ismail, Detecting sentiment from Bangla text using machine learning technique and feature analysis, *Int. J. Comput. Appl.* **153** (2016), 28–34.
- [22] B. G. Patra, D. Das, A. Das and R. Prasath, Shared task on sentiment analysis in Indian languages (SAIL) tweets—an overview, in: *International Conference on Mining Intelligence and Knowledge Exploration*, Hyderabad, India, Springer, pp. 650–655, 2015.
- [23] S. Phani, S. IIEST, S. Lahiri and A. Biswas, Sentiment Analysis of Tweets in Three Indian Languages, *WSSANLP 2016* **1001** (2016), 83–92.

- [24] S. Sachin Kumar, B. Premjith, M. Anand Kumar and K. P. Soman, AMRITA_CEN-NLP@ SAIL2015: sentiment analysis in Indian Language using regularized least square approach with randomized feature learning, in: *International Conference on Mining Intelligence and Knowledge Exploration*, Hyderabad, India, Springer, pp. 671–683, 2015.
- [25] M. Schuster and K. K. Paliwal, Bidirectional recurrent neural networks, *IEEE Trans. Signal Process.* **45** (1997), 2673–2681.
- [26] S. Se, R. Vinayakumar, M. Anand Kumar and K. P. Soman, AMRITA-CEN@ SAIL2015: sentiment analysis in Indian languages, in: *International Conference on Mining Intelligence and Knowledge Exploration*, Hyderabad, India, Springer, pp. 703–710, 2015.
- [27] S. Se, R. Vinayakumar, M. A. Kumar and K. P. Soman, Predicting the sentimental reviews in Tamil Movie using machine learning algorithms. *Indian J. Sci. Technol.* **9** (2016), 1–5.
- [28] S. Seshadri, A. K. Madasamy and S. K. Padannayil, Analyzing sentiment in Indian languages micro text using recurrent neural network, *IIOAB J.* **7** (2016), 313–318.
- [29] M. Sokolova and G. Lapalme, A systematic analysis of performance measures for classification tasks, *Inf. Process. Manag.* **45** (2009), 427–437.
- [30] X. Sun, C. Li and F. Ren, Sentiment analysis for Chinese microblog based on deep neural networks with convolutional extension features, *Neurocomputing* **210** (2016), 227–236.
- [31] V. Uma and N. Kausikaa, Sentiment analysis of English and Tamil tweets using path length similarity based word sense disambiguation, *IOSR J. (IOSR J. Comput. Eng.)* **1** (2016), 82–89.
- [32] J. Wang, L. C. Yu, K. R. Lai and X. Zhang, Dimensional sentiment analysis using a regional CNN-LSTM model, in: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Volume 2: Short Papers), Berlin, Germany, Vol. 2, pp. 225–230, 2016.
- [33] T. Wiatowski and H. Bölcskei, A mathematical theory of deep convolutional neural networks for feature extraction, *IEEE Trans. Inf. Theory* **64** (2017), 1845–1866.
- [34] G. Xun, Y. Li, W. X. Zhao, J. Gao and A. Zhang, A correlated topic model using word embeddings, in: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, Melbourne, Australia, 2017. doi: 10.24963/ijcai.2017/588.