Deep Neural Network Models for Emotion Classification on Twitter Data

Titli Sarkar
Department of Computer Science
University of Louisiana at Lafayette
Lafayette LA, USA
C00222141@louisiana.edu

Dr. Anthony S.Maida
Department of Computer Science
University of Louisiana at Lafayette
Lafayette LA, USA
asm6678@louisiana.edu

***Abstract*— Twitter is one of the most popular social media nowadays used for expression people's opinions. This paper addresses the problem of classifying emotion on tweets. Four basic types of emotions: anger, fear, joy, and sadness are identified. Four neural network models: LSTM, bidirectional LSTM, CNN-LSTM and GRU are used as to classify sequential data with semantic meaning. After preprocessing with sentence tokenize and stopwords removal, feature selection is done using three types of word embeddings, namely, GloVe[15], word2Vec[16] and Emoji2Vec[17]. The word embeddings of each tweet and class label of each of them are passed into the neural network model for training and testing purpose. Best predictive accuracy of 40% with LSTM, 41% with bidirectional LSTM, 39% with CNN-LSTM and 34% using GRU are achieved on test dataset. Different feature selection methods in combination is needed to be used and different neural network models must be considered to increase the accuracy of the model.**

Keywords *- Twitter Emotion Classification; GloVe; Word2Vec; Emoji2Vec; LSTM; BiLSTM; CNN-LSTM; Natural Language Processing*

I. INTRODUCTION

Emotions play key role in human intelligence, rational decision making, social interaction, perception, memory, learning, creativity, and more. Nowadays, social media provides a platform for business which allows to connect the owner with their customers to advertise or to know the customer's perspective of the products and services. Social media plays a significant role in politics as the crowd's reaction is one of the most important metric for the politicians to decide their future steps. Sentiment analysis is one of a most popular subtopic of natural language processing and text mining that deals with the automated discovery and extraction of knowledge about people's sentiments, evaluation and opinions from textual data such as personal blogs, review websites and customer feedback forms. Sentiment analysis has received significant interest in recent times because of its practical usage and application in today's environment. Twitter is one of the most popular social media for expressing emotions. The goal of Twitter emotion classification is to automatically determine the emotion expressed in a tweet. After analyzing the trend of the basic emotions present in popular tweets, we found out people have expressed four types of basic emotions: anger, fear, joy and sadness. We propose a novel method of classifying the emotion for tweets using LSTM, biLSTM, CNN-LSTM and GRU separately.

We are thankful to SemEval'2018 organizers for providing the data for the Task 1 which is Affect in tweets, falls under the category Affect and Creative Language in Tweets. The dataset is labelled and divided in three parts: Train Data, Development Data and Test Data. We have processed all the data in each of the train, development and test set before feeding them to the neural network as a mandatory part of any natural language processing problem. We have sentence tokenized and word tokenized each tweet, stripped punctuations and removed stop words from them using python NLTK library. After preprocessing is done, we have represented each word of the tweets as word embedding using combination of GloVe, Word2Vec and Emoji2Vec as feature selection methods. To make the tweet corpus equidimensional, zero padding is done for each tweet. The preprocessed corpus is now fed into a neural network. We have tried the models with single and multiple deep hidden layers and compared the results. The result shows that BiLSTM works slightly better than all other models.

This paper is organized as follows. In section II, the related work is reviewed. Section III describes the methods and ideas which includes the data preparation, feature selection steps. In section IV, experiments with different models and results are included. Section V gives a general analysis of results. Finally, in section VI, we have drawn the conclusion.

II. RELATED WORK

The sentiment can be found in the comments or tweet to provide useful indicators for many different purposes [1]. Also, [2] stated that a sentiment can be categorized into two groups, which is negative and positive words. Sentiment analysis is a natural language processing technique to quantify an expressed opinion or sentiment within a selection of tweets [3]. There exist two main approaches for extracting sentiment automatically which are the lexicon-based approach and machine-learning-based approach [4]. NLP techniques are based on machine learning and especially statistical learning which uses a general learning algorithm combined with a large sample, a corpus, of data to learn the rules [5].

Most of the work related to Twitter sentiment analysis can be divided into supervised methods [6] and lexicon-based methods. Supervised methods are based on training classifiers (such as Naive Bayes, Support Vector Machine, Random Forest) using various combinations of features such as Part-Of-

Speech (POS) tags, word N-grams, and tweet context information features, such as hashtags, retweets, emoticon, capital words etc. Lexicon-based methods determine the overall sentiment tendency of a given text by utilizing pre-established lexicons of words weighted with their sentiment orientations, such as SentiWordNet [7]. Researched has noticed that the sentiment of a tweet is implicitly associated with the semantics of its context. Therefore, implicit semantics in the words of tweets should be considered. Deep learning methods are now well established in machine learning [8]. Kim [9] studied the performance of convolution neural network for sentence level sentiment classification tasks on a pre-trained word vector model. Experiments on different datasets show that convolution neural networks can improve classification performance. Johnson et al.[10] proposed a model for high dimensional text for text classification and obtain several state-of-the-art performances on some benchmark data sets for sentiment categorization but swapped in high dimensional 'one-hot' vector representations of words as CNN inputs. Their focus was on classification of longer texts, rather than sentences. Tang et al. [11] proposed a method of continuous speech using neural network automatic acquisition of word emotion information representation, building micro-blog text emotion feature vector, excellent performance in SemEval2013 classification tasks.
Taner Danisman et.al [12] studied the automatic classification of anger, disgust, fear, joy and sad emotions in text. The study was conducted on ISEAR (International Survey on Emotion Antecedents and Reactions) dataset. For the classification they have used Vector Space Model with a total of 801 news headlines provided by "Affective Task" in SemEval 2007 workshop which focuses on classification of emotions and valences in text. They have compared their results with ConceptNet and powerful text-based classifiers including Naive Bayes and Support Vector Machines. Their experiments showed that VSM classification gives better performance than ConceptNet, Naive Bayes and SVM based classifiers for emotion detection in sentences. An overall F-measure value of 32.22% and kappa value of 0.18 for five class emotional text classification on SemEval dataset which is better than Navie Bayes (28.52%), SVM (28.6%) was obtained.

The Sentiment Analysis in Twitter task has been run yearly at SemEval since 2013 [13]. In the latest work, Maximilian Koper et al. in EmoInt 2017 [14] WASSA-2017 shared task on the prediction of emotion intensity in tweets attempts to come up with baseline features which is combination of SparseFeatures which refer to word and character n-grams from tweets, LexiconFeatures which are taken from several emotion and sentiment lists and the EmbeddingsFeatures before feeding them to a recurrent neural network.

III. Methods and Ideas

In this work, we present four different kinds of models for comparison such as LSTM, BiLSTM, CNN-LSTM and GRU which determine the emotion expressed in a tweet, given an arbitrary tweet. We get the output of the neural network in four output neurons and select the neuron with highest score as the output emotion. This work can be extended to predict emotion intensity also if we train the model with emotion labels as well as intensity scores.

A. DataSet

The dataset we have picked for usage from EmoInt 2018 task, have a total 8566 tweets in English language for training and validation. We have picked total 2000 mixed tweets from each of the anger, fear, joy and sadness testing dataset in English language to use as our test data. We have made sure the dataset is equibalanced in each type of emotions by picking 500 tweets for each emotion testing data. Each of our datasets are divided in three parts: training data, development data and test data. In each of the set, we have separate files for anger, fear, joy and sadness. Each of the files has labelled data in the form "ID, Tweet, Affect Dimension, Intensity Score". The field Affect Dimension has four values: *anger, fear, joy, sadness.* We have annotated or labelled the emotions as anger - 1, fear - 2, joy - 3, sadness - 3 and combined the development data with the training data together to be used as training data. The reason of including the development data in training data is to give the model a little more data for learning such that it can make better prediction on test data. As our task is classification task, we have combined tweets for all the emotions together in a single file for each of training and testing purpose.

B. Preprocessing

Preprocessing is a crucial step in natural language processing. Our data is available in raw form. It need to be processed in a form that is free from noise and hence does not create any problem for finding the word embedding before feeding into the neural network. Following is the sequence of steps:

1. *Tokenization:*

Tokenization is the process of replacing data with unique identification symbols that gets rid of the noise but retain all the essential information about the data without compromising its meaning. It is usual practice in natural language processing and machine learning that we should split each paragraph into sentences and each sentence into words. We have used python NLTK tokenizer to break tweets which have more than one sentence to individual sentences and break each individual sentence into individual words.

2. *Removing URL's:*

Tweets often contain links to other websites. This is the case when these websites contain content, referencing which the tweet was posted. These are just unwanted noise and contribute in no positive way to classification and hence must be removed. This architecture uses regular expressions to remove any URL that is present in tweets.

3. *Removing @'s:*

The sign @ allows users to mention other users in their tweets. Like "@carljones very happy to hear your story!" tweet mention "carljones" which does not have any useful meaning to emotions, henceforth is removed.

4. *Removing hashtags:*

Sometimes people use normal words preceded by hashtags in sentences. For example, "Finally time to eat some #cake!". These sentences when tokenized are broken down along with the hashtag, and the words "cake" and "#cake" have different word representations. One must remove the hashtag punctuation symbol and interpret it as a plain word. This again is done by using regular expressions.

5. *Strip punctuations:*

We are handing with emotions; therefore, it is usual that the tweets will have exclamation marks (!) and question marks (?). We have removed punctuations using python NLTK library in built method.

6. *Removing Numbers:*

Numbers does not have any importance when detecting emotions in text. In most cases, they simply convey some time or date or quantity and are totally irrelevant for in the current domain. They simply add noise and hence must be removed. Again regular expressions are employed to replace them with blanks.

7. *Removing Stop Words:*

Stop words are a set of commonly occurring words in any language. They typically are prepositions, conjunctions and determiners like “the”, “a”, “an”, etc. Python NLTK library is used to remove stop words.

8. *Lowercase Conversion:*

The last step in pre-processing is to convert all the text to lowercase. It is important because it ensures that the word vector of each word is not affected by their case. This helps maintain uniformity throughout in the next step, where we are finding word embedding of each words in the tweet.

C. Feature Selection

We are working with textual sequential data which have some semantics, therefore, we need to retain the semantics of the data in our representation. We can use pre-trained models which have been trained on very large text corpus which have been derived from various sources like news, Twitter, Wikipedia etc. Advantage of using a custom trained model is that they learn words in the domain of application. For example, if one trains a model based on text corpus from Twitter, words will be learned in the domain of Twitter. Nowadays, a word embedding model called GloVe [15], developed by Standford NLP group is the most popular word embedding model for word representation for Global Vectors, because the global corpus statistics are captured directly by the model. The classic Word2Vec [16] model implemented with skip-gram is also used to obtain a dense, low-dimensional and real-valued vectors for words. Emojis play a significant role in special type of applications like twitter emotion analysis. We have adopted a pretrained embeddings for all Unicode emojis which are learned from their description in the Unicode emoji standard emoji2vec[17]. Once all the models for word embeddings are acquired, we have used GloVe+word2vec+emoji2vec to generate word embeddings for each of the tweets of training, development and test data set which are cleaned by several preprocessing steps, as a vector of features. This is done as follows:

1. *Loading Models:*

GloVe: It is the newest and best suited model for twitter data till date. We have downloaded the pre-trained GloVe model for Twitter with embedding length 100. Those pre-trained was converted to word2vc format for using in our model using python genism library. Then, using the same module, we have loaded the pre-trained model in word2vec format.

Word2Vec: It is shown in the literature that training a word to vector model with domain specific data helps to achieve best prediction. Therefore, we have trained a word2vec model [16] implemented with skip-gram using a corpus of 80000 tweets from Semeval 2017 Task1 train+test+dev dataset. The dimension of each word embedding is 100.

Emoji2Vec: Emojis play a vital role in understanding the emotion in tweets. We have used a pretrained model of emoji to vector for generating emoji embedding of length 300 [17].

2. *Finding Word Embeddings:*

After the preprocessing step, we have broken each tweet into a list of separate word tokens. Now we find three embedding for each token using the GloVe (length=100), Word2Vec(length=100) and Emoji2Vec(length=300). Therefore, each token has an embedding vector of length 100+100+300=500. But, all tokens do not have representation in all the above three models, therefore, if a token do not have representation in any model, the corresponding word embedding is filled with zeros of equal length.

3. *Making Corpus Equidimensional:*

All tweets are not of same length in corpus. We must make the corpus equidimensional by adding the zero padding with maxlen=length of the longest tweet using Keras.

4. *One Hot Encoding:*

Each label is one hot encoded in vector of length equal to the number of total labels. For categorical variables where no ordering among the variable values exists, one hot encoding is needed instead of integer encoding. Using integer encoding means we are allowing the model to assume a natural ordering between categories and it may result in deficient performance. In this case, a one-hot encoding can be applied to the integer representation where each label is represented as a binary vector of 0 and 1 of length equal to total number of labels or classes and the vector has value 1 only at the index equal to the integer value. For example, if we have 10 classes or labels, then class 4 is represented by a vector [0 0 0 0 1 0 0 0 0 0].

At this stage, we have equidimensional corpus for training and one-hot encoded labels. For custom trained vectors, the models are trained using nearly 9000 tweets. They will be tested against pretrained vectors of dimension 500 for 2000 test data which includes tweets of all kinds of emotions in a balanced manner. We have created a train dataset by concatenating the word embedding of the tweets present in the training dataset and the word embedding of the tweets present in the development dataset. Similarly, we have created a train label set by concatenating the emotion annotated labels in the train dataset and the emotion annotated labels in the development dataset. These two are served as training corpus and training labels for the neural network models.

IV. Experiments and Results

We have use LSTM and bidirectional LSTM as our models for building classifier for this specific problem of Twitter emotion classification.

a. LSTM

- *Methods:*

Our first model is Long Short-Term Memory or LSTM. The main reason of selecting LSTM is the nature of our data is sequential with semantic meaning. Our deep neural network needs to remember some long-term past context and need to erase some information which is not relevant to future prediction. Recurrent neural network does not work good in situations where we need to preserve long-term dependency because it has the drawback of vanishing or exploding gradient.

The information goes in repetitive loops in training phase of RNN which results in very large updates to neural network model weights. By repeatedly multiplying the gradient (partial derivative with respect to the inputs which decides how much output will vary with a little change of input) which have values >1, if the weights become so large that they overflow and result NaN values by exponential growth of accumulated error gradients, the explosion of gradient occurs. By similar process, if the gradient values are <1, the gradient tends to vanish and gives rise to the vanishing gradient problem. LSTM can overcome this problem by using gates to control the memorizing process; each LSTM cell has one forget get for forgetting irrelevant information and memorizing relevant information and one output gate to produce output to next time step. The LSTM architecture is briefly described in Figure 1.

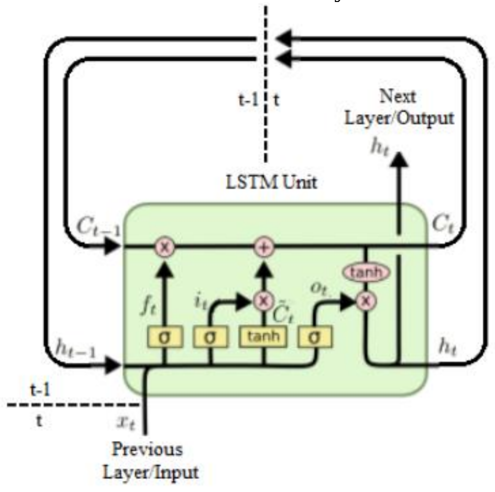


Figure 1: Architecture of a LSTM cell

In Figure 1, the symbol *(a)* X represents scaling of information, *(b)* + represents the addition of information, *(c)* σ represents sigmoid layer, *(d)* tanh represents tanh layer, (e) h_{t-1} represents output of LSTM unit from last time step, *(f)* c_{t-1} represents memory of LSTM unit from last time step, *(g)* x_t represents current input, *(h)* c_t represents new updated memory and (*i)* h_t represents current output. The second derivative of tanh function can sustain for a long range before going to zero, therefore it solves the vanishing gradient problem. Sigmoid can output 0 or 1, therefore, it can be used to decide which part of input information to forget and which part to remember.

The input information passes through many such LSTM units before giving final output. The main components of an LSTM units as shown in Figure 1 are:

i. Forget Gate:

This gate can forget the unnecessary information. It is indicated by f_t from the left in the diagram. The input x_t and h_{t-1} are added and fed into the sigmoid layer which decides which parts from old output should be removed (by outputting a 0). The output of this gate is given by f_t *c_{t-1}. The value of f_t is given by

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

where σ denotes the sigmoid layer, at timestep t, the dot product of weight matrix W_f and the multiplication result of hidden layer h_{t-1} in timestep t-1 and input x_t is added with a bias b_f and outputted as f_t.

ii. Input Gate:

The process of adding some new information to the cell state can be done via the input gate which is indicated by i_t in the diagram. This addition of information is basically two-step process as seen from the diagram above: *a.* regulate what values need to be added to the cell state by filtering through a sigmoid function (output values range from 0 to 1), *b.* generate a vector containing all possible values that can be added to the cell state using the tanh function (output values range from -1 to +1). The output of this gate are given by

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

This step is for deciding what new information we're going to store in the cell state: a sigmoid layer called the "input gate layer" decides which values to keep as update and then a tanh layer creates a vector of new candidate values c_t hat, that could be added to the state. In the next step, these two are combined to create an update to the state.

iii. Memory Cell:

The output of the sigmoid function and the tanh function gets multiplied and this new memory then gets added to old memory c_{t-1} to give c_t to update the new cell. This process is indicated by the X and + symbols after i_t, which leads to c_t. After this stage, only that information is added to the cell state that is important and is not redundant. The output of this state is given by

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

Now the old cell state, c_{t-1} will actually be updated into the new cell state c_t after forgetting the unnecessary things.

iv. *Output Gate:*

The last part is to decide the output from this cell. The diagram has o_t to indicate the operation of this gate. The functioning of an output gate can be broken down to three steps: *a.* create a vector after applying tanh function to the cell state, thereby scaling the values to the range -1 to +1, *b.* make a filter using the values of h_{t-1} and x_t, such that it can regulate the values that need to be output from the vector created in step *a.* by using a sigmoid function and *c.* multiply the value of this regulatory filter to the vector created in step *a.* and send it out as a output and also to the hidden state of the next cell. The mathematical formula is as follows:

$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

The output of the LSTM cell will be based on the current updated cell state. First, a sigmoid layer decides what parts of the cell state are going to the output and then the cell state is passed through tanh (to push the values to be between −1 and 1) and multiplied by the output of the sigmoid gate ensuring that only output the parts which are decided will be outputted.

Settings: We have chosen to use Keras neural network API in Python for easy and fast prototyping. One LSTM layer of 64 units is used. The input is word embedding with each word vector length 500, therefore, the input dimension is (#of word embedded tweet vectors X 500). The inputs are fed into a LSTM layer with 64 units and the output of the 64 units of the LSTM layer is fed into an output layer of 4 units, as we have 4 different type of class labels for *anger, fear, joy* and *sadness.* The number of hidden units in LSTM layer was changed while doing hyperparameter tuning in experiments. All the input word embedded vectors are passed through the LSTM units in a connected fashion.

- *Results:*

We have tested all our models on 2000 test data and nearly 8500 train+validation data for all the models. We have fitted each model 20 times and predicted the emotion labels on the test data. We have acquired F-score, Recall and Accuracy as evaluation measures. F-score intuitively tells us how precise our classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances). Recall gives an estimate of how many correct instances are retrieved. We have also computed the confusion matrix on the final predicted result in comparison to the actual emotion labels on the original test dataset. We have calculated pointwise difference between the actual and predicted labels and plotted the differences as a graph.

1. *Evaluation of LSTM:*

We have varied the all the hyperparameters and the result of running the experiments with best parameter settings are listed below:

Model	**Batch Size**	**Epoch**	**Test Accuracy**	**F-score**	**Recall**
LSTM	**128**	5	0.38	0.36	0.37
	128	20	0.40	0.37	0.38
	128	80	0.37	0.36	0.37

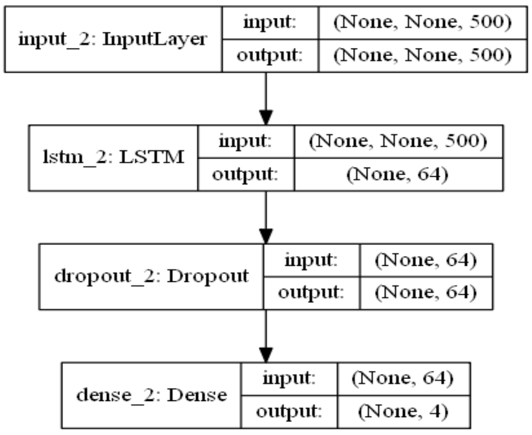
2. *Confusion Matrix:*

The confusion matrix from the last run is given below:

		PREDICTED			
		ANGER	**FEAR**	**JOY**	**SADNESS**
ACTUAL	**ANGER**	189	153	53	105
	FEAR	185	161	53	102
	JOY	100	25	290	86
	SADNESS	142	51	44	237

3. *Model Plot:*

The LSTM model flow is shown below:



b. **biLSTM:**

- *Methods:*

LSTM could not achieve much high accuracy, therefore we have tried bidirectional LSTM as our second model which preserves the context from future as well as past while unidirectional LSTM only preserves past context at any point of time. biLSTM is particularly useful in the applications where data is sequential, and we need some future context to understand the semantic meaning od a sentence. It has two networks, one access information in forward direction and another access in the reverse direction (as shown in Figure 2 below). These networks have access to the past as well as the future information and hence the output is generated from both the past and future context. The basic building block of forward and backward LSTM in each time step are not connected in order to preserve the effect of forward and backward context. Bidirectional LSTMs are supported in python Keras via the Bidirectional layer wrapper.

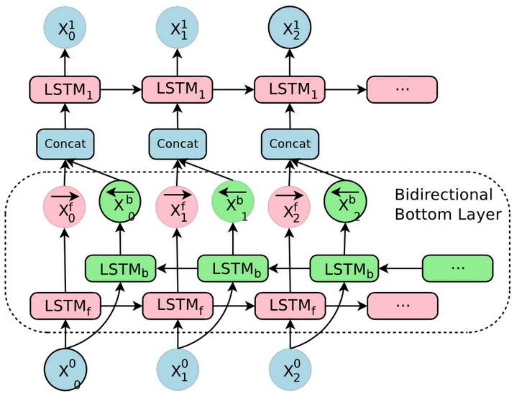


Figure 2: Architecture of a BiLSTM layer

We have used Keras bidirectional module Bidirectional(LSTM()) with output dimension of 64 for each of the forward and backward layers. The input embeddings of each token in a sentence are fed into a bidirectional LSTM in each timestep in an unrolled network. The output of each forward and backward equivalent unit are concatenated and fed into the LSTM units in next time step in case of more than one layer and to the output in case of single layer network. The output of the 64 units of the BiLSTM layer is fed into an output layer of 4 units, as we have 4 different type of class labels for *anger*, *fear*, *joy* and *sadness.*

Settings: The bidirectional LSTM has input dimension of 500 (the size of embedding vectors) and output dimension of 64 (i.e. the size of the vectors from LSTM output layer). Those outputs are fed into an output layer having #of class=4 units interpreted for for four kind of emotions. We have used a single layer biLSTM with *dropout* of 0.3 *softmax* as output layer activation, *categorical cross entropy* as loss function and *adam* optimizer with learning rate 0.001 in 20 epochs and batch size 128.

- *Results:*

2. *Evaluation of biLSTM:*

We have varied the all the hyperparameters and the result of running the experiments with best parameter settings are listed below:

Model	**Batch Size**	**Epoch**	**Test Accu**	**F-score**	**Recall**
biLSTM	**128**	5	0.38	0.36	0.37
	128	20	0.41	0.39	0.38
	128	80	0.39	0.37	0.38

2. *Model Plot:*

The biLSTM model plot is similar as LSTM.

3. *Confusion Matrix:*

The confusion matrix from the last run is given below:

		PREDICTED			
		ANGER	**FEAR**	**JOY**	**SADNESS**
ACTUAL	**ANGER**	183	156	56	105
	FEAR	285	61	53	102
	JOY	101	25	289	86
	SADNESS	174	52	64	208

c. CNN-LSTM

- *Methods:*

The CNN architecture is used to extract best features from the input dataset and the LSTM architecture is used to learn the sequential data. The architecture is shown in the diagram below. First, we obtain embedding for each token in each sentence in the corpus and we feed them to the CNN layer. The output of the CNN is a stack of best feature set in reduced dimensional space and the flatten output layer is fed into a fully connected layer of LSTM. The LSTM memory cells preserve the long-range dependency among the tokens, therefore best suited for capturing the context or semantic meaning in the data.

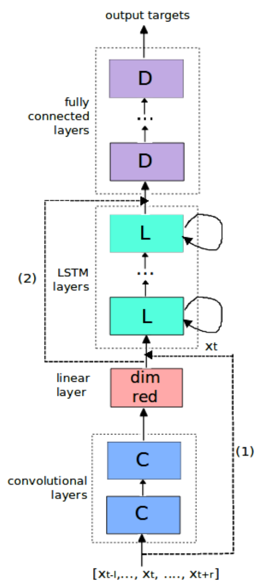


Fig 3: a CNN-LSTM general architecture with two convolution layers and two LSTM layers

Settings: A 1D convolutional neural network comprised of Conv1D and MaxPooling1D layers is used. The Conv1D will interpret snapshots of the image (e.g. small squares) and the polling layers will consolidate or abstract the interpretation for extracting features and make the network deeper more cheaply. The convolution layer maps the input layer to a stack of 1024 output channels. The flatten output layer will transform each convolution layer to a vector ready to be fed as input to LSTM layer and maps to a layer of 64 LSTM hidden units. The hidden layer is connected to an output layer with 4 output nodes for four kind of emotions. We have used a single layer LSTM with *dropout* of 0.5, *tanh* as convolution activation, *softmax* as LSTM activation, *categorical crossentropy* as loss function and *adam* optimizer with learning rate 0.001 in 80 epochs and batch size 32.

- *Results:*

1. *Evaluation of CNN-LSTM:*

We have varied the all the hyperparameters and the result of running the experiments with best parameter settings are listed below:

Model	**Batch Size**	**Epoch**	**Test Accu**	**F-score**	**Recall**
CNN-LSTM	**128**	10	0.39	0.33	0.32
	128	20	0.40	0.33	0.32
	32	80	0.31	0.33	0.31

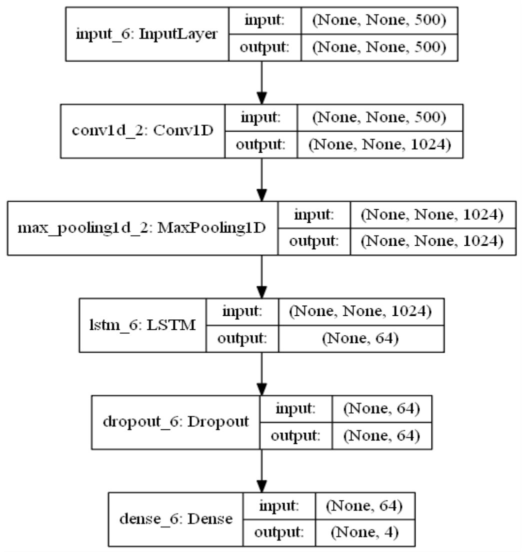
2. *Confusion Matrix:*

The confusion matrix from the last run is given below:

		PREDICTED			
		ANGER	**FEAR**	**JOY**	**SADNESS**
ACTUAL	**ANGER**	124	220	55	101
	FEAR	286	79	35	101
	JOY	124	63	218	96
	SADNESS	153	84	62	199

3. *Model Plot:*

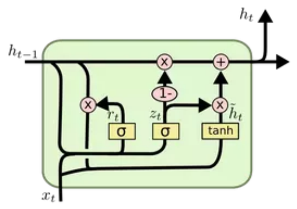
The CNN-LSTM model flow is shown below:



d. GRU

- *Methods:*

Gated Recurrent Unit or GRU is a recurrent neural network similar to LSTM but utilizes different way of gating information i.e. does not have a cell state and has 2 gates (update, reset) instead of 3(forget, update, output) to prevent vanishing gradient problem. The update gate decides on how much of information from the past should be let through and the reset gate decides on how much of information from the past should be discarded.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Fig 4: a GRU RNN cell

In the above figure, z_t represents the update gate operation using a sigmoid function which decide on what values to let through from the past. h_t represents the reset gate operation, where we multiply the concatenated values from the previous time step and current time step with r_t. This produces the values that we would like to discard from the previous time steps.

The GRU unit controls the flow of information like the LSTM unit, but without having to use a *memory unit*. It just exposes the full hidden content without any control. It is computationally more efficient than LSTM in terms of simpler structure and less memory usage but comes next to LSTM in terms of performance. GRU can be used when we need to train faster, don't have much computation power and do not need to capture long-distance relations.

Settings: The GRU has input dimension of 500 (the size of embedding vectors) and output dimension of 64 (i.e. the size of the vectors from GRU output layer) with dropout 0.3.
We have added another layer with 256 hidden units,*dropout* of 0.3*softmax* as output layer activation Those outputs are fed into an output layer having #of class=4 units interpreted for for four kind of emotions. We have added a layer with with *dropout* of 0.3 *softmax* as output layer activation, *categorical cross entropy* as loss function and *adam* optimizer with learning rate 0.001 in 20 epochs and batch size 128.

- *Results:*

1. *Evaluation of GRU:*

We have varied the all the hyperparameters and the result of running the experiments with best parameter settings are listed below:

Model	**Batch Size**	**Epoch**	**Test Accu**	**F-score**	**Recall**
GRU	**128**	10	0.34	0.33	0.32
	128	20	0.34	0.33	0.32
	32	80	0.31	0.33	0.31

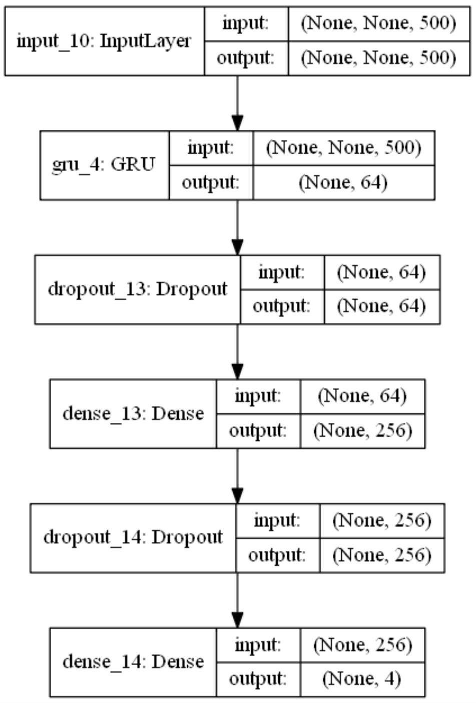
2. *Confusion Matrix:*

The confusion matrix from the last run is given below:

		PREDICTED			
		ANGER	**FEAR**	**JOY**	**SADNESS**
ACTUAL	**ANGER**	213	121	34	132
	FEAR	70	258	35	138
	JOY	37	88	265	111
	SADNESS	71	128	54	245

3. *Model Plot:*

The GRU model flow is shown below:



V. Analysys of Results

A. *Accuracy Comparison with Parameter Variation*

We have used high-level neural networks API Keras in Python 3 in Anaconda to build the models and testing them. We have executed the code for each model for 10 to 80 times and took the best result in terms of accuracy on test data. All the models are tried with different parameters and hyperparameters like different number of neurons in hidden layers, different hidden layers, different activation function, loss function, dropout value, optimizers, kernel size and different epochs. The best predictive accuracy on test data of each model with best settings are compared in table1.

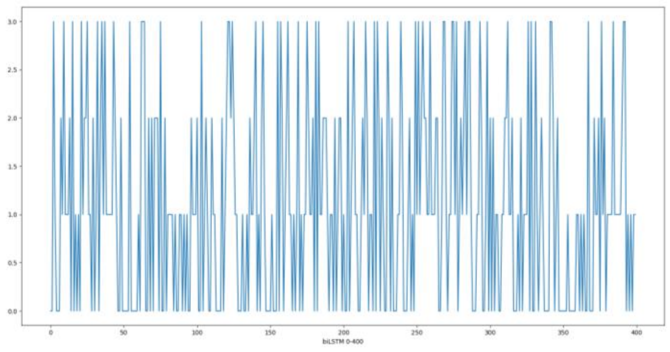
Model	#epoch	#train example	#test example	Model Accuracy	Test Accuracy
LSTM	20	8566	2000	0.80	0.40
Bidir LSTM	20	8566	2000	0.84	0.41
CNN-LSTM	20	8566	2000	0.78	0.39
GRU	20	8566	2000	0.72	0.34

Table1: Accuracy Comparison with different parameters

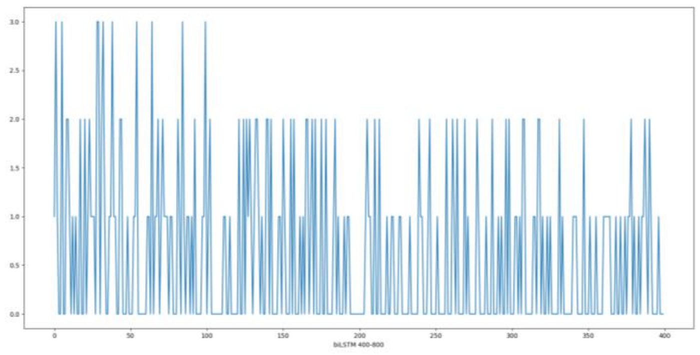
The increase in #of iterations give better result than increasing the #of last hidden layer neurons in case of LSTM, but if we increase the iteration too much, the accuracy drops. Best test accuracy was obtained with 20 epochs. Dropout does not affect much in the accuracy. However, the change of activation function has affected the accuracy significantly. We could clearly see that *softmax* as activation function works much better than *reLu.* We assume that happen because of nature of our data.

B. *Plot of Variation:*

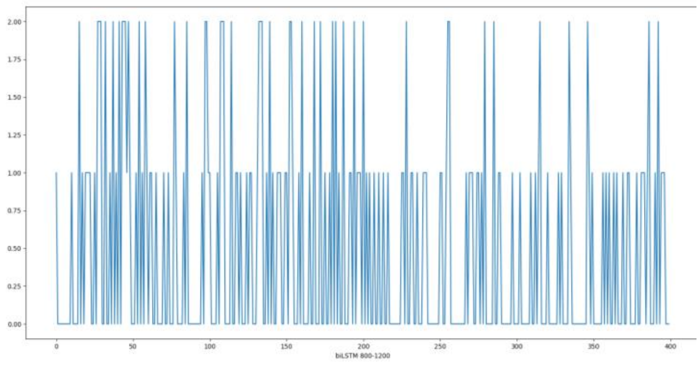
The actual labels for test data make a one-dimensional vector of 2000 labels belonging to any of the *anger, fear, joy* or *sadness* classes. The predicted labels also produce a similar one-dimensional vector of length 2000. The elementwise difference of predicted test labels and actual test labels gives us a one-dimensional vector of length 2000. Each value v_i in variation vector is equal to $|predicted_i - actual_i|$ where $1 \leq i \leq 2000$. We have plotted the variation vector in graph. The number of our test instances are huge to fit the resultant graph in this paper in an interpretable manner. Therefore, we have divided our variation vector in five parts with 1/5th of the total data in each part as follows: a. first 1-400 data points b. next 400-800 data points c. next 800-1200 data points, d. next 1200-1600 data points and e. next 1600-2000 data points and plotted them in graphs all the models. The result of one model (biLSTM) is shown in figure 5.



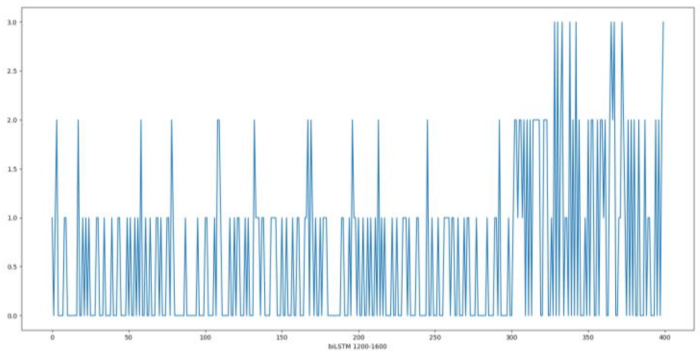
a.



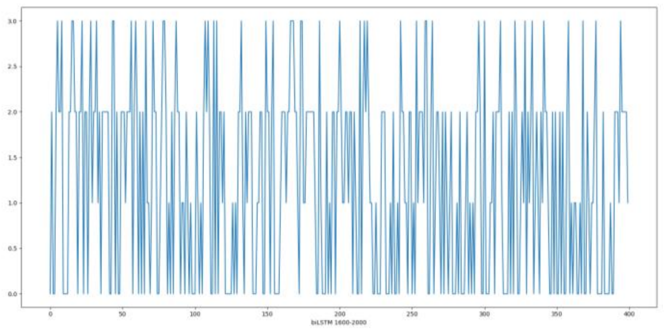
b.



c.



d.



e.

Figure 5: Elementwise difference of predicted and actual labels for biLSTM: a. first 0-400 points b. second 400-800 points c. third 800-1200 points d. fourth 1200-1600 points e. last 1600-2000 points.

C. *Plot of Error Bar:*

We have used Python in-build modules for creating models and they have in-build method for randomly initialization of weight matrices. Therefore, in each iteration, the weight initialization differs and the accuracy differs. We have plotted the variability of accuracy values in error bar graphs to indicate the *error* or variation in measurements of different iterations. Figure 6 shows the variation in the Model errors and Figure 7 shows the variation in the test errors from 20 trial execution of the code. The mean, standard deviation and maximum and minimum range of values are indicated in the graphs.

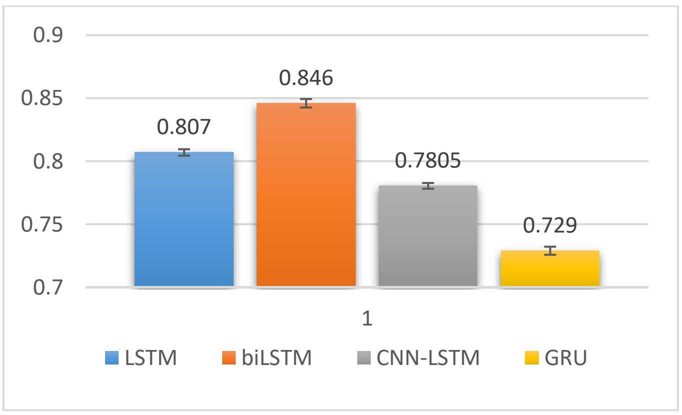


Figure 6: Error bar for Model error (20 executions)

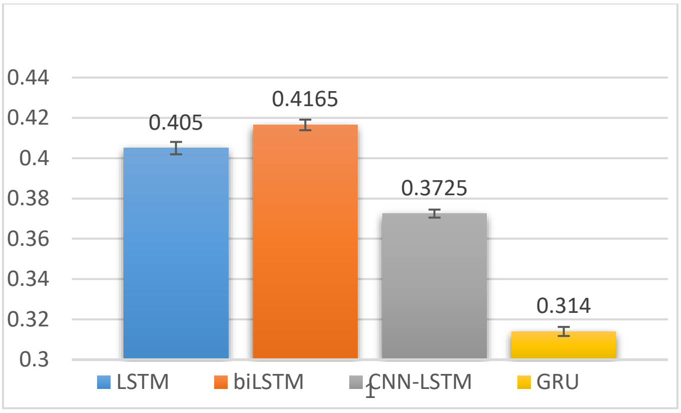


Figure 7: Error bar for Test error (20 executions)

The error bar with 95% confidence interval (i.e. a range of values of standard deviation which gives 95% certainty of containing the true mean of the values) is shown by the black lines in the bar chart.

VI. Future Work and Conclusion

This project addressed the problem of identifying emotions in tweets. This is a specific kind of text mining problem. We could achieve best accuracy of 41% using single layer biLSTM. The performance of our models is lesser than the literature in terms of F-score and Recall. We can think the following reasons: (i) The insufficient training samples for word2vec model; most of the literature has trained the model with at least 3M tweets which requires huge time and powerful resources, (ii) The high dimension of the embeddings; maximum dimension for word embeddings used in the literature is 250, where we has to use 100(GloVe)+100(word2Vec)+300(emoji2vec)=500 dimensions, as we have used pretrained vectors for GloVe and emoji2vec. If we can train our word embedding models, we can set the dimensions as per requirement. (iii) In preprocessing step: we did not consider effect of negative words which can change the meaning of word to opposite polarity if prefixed to a word, we did not consider the special characters like (?, !) which can add some emotion value in the tweets, we did not consider hashtags(#) which may add some value in special kind of tweets like hate speech etc. In the related work, we have seen that helps in improving accuracy by 10-20%. One other direction is to introduce attention in our RNN, which sounds reasonable to impose more weightage to important words in a sentence. We also want to consider the Psycholinguistic which is being used recently to classify texts and sentences for a variety of tasks.

In future, we want to consider Lexicon embeddings and use of attention mechanism in our models. Moreover, we want to design an ensemble neural network model to classify tweeter emotions and predict the emotion intensity of the classified tweets.

References

[1] M. Annett, and G. Kondrak, “A Comparison of Sentiment Analysis Techniques: Polarizing Movie Blogs,” Conference on web search and web data mining (WSDM). University of Alberia: Department of Computing Science, 2009.

[2] H. Saif, Y.He, and H. Alani, “SemanticSentimentAnalysisof Twitter,” Proceeding of the Workshop on Information Extraction and Entity Analytics on Social Media Data. United Kingdom: Knowledge Media Institute, 2011.

[3] T. Carpenter, and T. Way, “Tracking Sentiment Analysis through Twitter,”. ACM computer survey. Villanova:VillanovaUniversity, 2010.

[4] E. Kouloumpis, T. Wilson, and J. Moore, “Twitter Sentiment Analysis:The Good the Bad and theOMG!”, (Vol.5). International AAAI, 2011.

[5] A. Blom and S. Thorsen, “Automatic Twitter replies with Python,” International conference “Dialog 2012”.

[6] Y Saif, H., He, Y. & Alani, H., Semantic sentiment analysis of twitter. The Semantic Web–ISWC 2012, pp.508-524,2012.

[7] Baccianella S, Esuli A & Sebastiani F. "Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining". In Proc. the Seventh conference on International Language Resources and Evaluation, Malta. Retrieved May. Valletta, Malta, pp.2200-2204, 2010.

[8] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. Nature,521(7553):pp.436–444, 2015.

[9] Kim, Y. Convolutional Neural Networks for Sentence Classification. Paper presented at the Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), october 25-29, Doha, Qatar,2014, pp.1746–1751, 2014.

[10] Rie Johnson and Tong Zhang. Semi-supervised convolutional neural networks for text categorization via region embedding. In Advances in Neural Information Processing Systems, pp.919–927,2015.

[11] Tang, D., FuruWei, Yang, N., Zhou, M., Liu, T., & Qin, B.. Learning Sentiment-Specific Word Embedding for Twitter Sentiment Classification. In the Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics Baltimore, Maryland, USA, June 23-25 2014, pp. 1555-1565,2014.

[12] Taner Danisman, Adil Alpkocak, “Feeler: Emotion Classification of Text Using Vector Space Model”.

[13] Preslav Nakov, Sara Rosenthal, Svetlana Kiritchenko, Saif M. Mohammad, Zornitsa Kozareva, Alan Ritter, Veselin Stoyanov, and Xiaodan Zhu. 2016b. Developing a successful SemEval task in sentiment analysis of Twitter and other social media texts. Language Resources and Evaluation 50(1):35–65.

[14] Maximilian Koper, Evgeny Kim ¨ and Roman Klinger, IMS at EmoInt-2017.Emotion Intensity Prediction with Affective Norms, Automatically Extended Resources and Deep Learning Institut fur Maschinelle Sprachverarbeitung ¨ Universitat Stuttgart, Germany

[15] Jeffrey Pennington, Richard Socher, Christopher D. Manning, Standford NLP group.GloVe: Global Vectors for Word Representation.

[16] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado and Jeffry Dean, “Distributed Representations of Words and Phrases and their Compositionality”, Proceeing of NIPS 2013

[17] Ben Eisner, Rocktaschel, Isabelle Augenstein, Matko Bosnjak and Sebastian Riedel, “emoji2vec: Learning emoji Representation form their Descriptions”, arXiv 2016.

Supplymentary Materials

The source code is available at following link: *https://github.com/TitliSarkar/Twitter-Emotion-Classify-CSCE-598*