

# Contents

Abstract

Acknowledgement

|                                                   |           |
|---------------------------------------------------|-----------|
| <b>1. Introduction</b>                            | <b>1</b>  |
| 1.1 What is Code Mixing? .....                    | 1         |
| 1.2 Sentiment Analysis .....                      | 1         |
| 1.3 Literature Review .....                       | 3         |
| 1.4 What I focused on .....                       | 5         |
| <b>2. Problem Statement</b>                       | <b>6</b>  |
| 2.1 Problem Definition .....                      | 6         |
| <b>3. Existing Methods and Limitations</b>        | <b>7</b>  |
| 3.1 Character-Level Models .....                  | 7         |
| 3.1.1 Introduction .....                          | 7         |
| 3.1.2 Limitations of Character-Level models ..... | 10        |
| 3.2 Sub-Word Level Representations .....          | 11        |
| 3.2.1 Introduction .....                          | 11        |
| 3.2.2 Hypothesis .....                            | 11        |
| 3.2.3 Methodology .....                           | 11        |
| 3.2.4 Model Architecture .....                    | 12        |
| 3.2.5 Limitations of Sub-Word level models .....  | 14        |
| <b>4. Dataset</b>                                 | <b>15</b> |
| 4.1 Some Major Data Issues .....                  | 15        |

|           |                                                      |           |
|-----------|------------------------------------------------------|-----------|
| 4.2       | Data Preprocessing .....                             | 17        |
| 4.3       | Words To Embeddings .....                            | 19        |
| 4.3.1     | GloVe Embeddings .....                               | 20        |
| 4.4       | Using GloVe Embeddings .....                         | 24        |
| <b>5.</b> | <b>Model Overview</b>                                | <b>25</b> |
| 5.1       | Introduction .....                                   | 25        |
| 5.1.1     | Basic RNN Learning Cell .....                        | 26        |
| 5.2       | Objective ..                                         | 28        |
| 5.2.1     | Methodology .....                                    | 28        |
| 5.3       | Architecture .....                                   | 30        |
| 5.4       | Experimental Setup .....                             | 31        |
| 5.5       | Experimental Setup .....                             | 32        |
| 5.6       | Method Suitability .....                             | 35        |
| <b>6.</b> | <b>Observation</b>                                   | <b>36</b> |
| 6.1       | Plot for Training/Validation Loss .....              | 37        |
| 6.2       | Plot for Training/Validation Accuracy .....          | 37        |
| 6.3       | Comparing Results .....                              | 38        |
| 6.4       | Validation of Proposed Hypothesis .....              | 39        |
| <b>7.</b> | <b>Conclusion</b>                                    | <b>40</b> |
| 7.1       | Future Work .....                                    | 40        |
|           | <b>References</b>                                    | <b>41</b> |
|           | <b>Appendix</b>                                      | <b>42</b> |
|           | Code for sentiment analysis of Code-Mixed text ..... | 42        |
|           | Code for testing the model .....                     | 48        |

# List of Figures

|     |                                                                        |     |
|-----|------------------------------------------------------------------------|-----|
| 1.1 | Proposed model Architecture                                            | 5   |
| 3.1 | Typical Character-level model for Sentiment Analysis                   | 8/9 |
| 3.2 | Schematic overview of the model Architecture                           | 13  |
| 3.3 | Illustration of the proposed methodology                               | 14  |
| 4.1 | Comments collected from the Facebook page (Before Translation)         | 17  |
| 4.2 | Example script of Google Translation API                               | 18  |
| 4.3 | Comments generated using Google Translation API (After Translation)    | 18  |
| 4.4 | Vector representation of a word using Word Embeddings                  | 19  |
| 4.5 | An Example 100-Dimensional embedding vector of GloVe<br>Word Embedding | 23  |
| 4.6 | Creation of Embedding vector from translated words                     | 24  |
| 5.1 | Representation of a time sequence RNN model                            | 25  |
| 5.2 | General architecture of an LSTM cell                                   | 26  |
| 5.3 | Mathematical computation of a single time step in the LSTM cell        | 27  |
| 5.4 | General architecture of a GRU cell                                     | 27  |
| 5.5 | Mathematical computation at a single time step in the GRU cell         | 28  |
| 5.6 | Instance of an LSTM cell at time step ‘t’                              | 29  |
| 5.7 | Illustration of the proposed methodology                               | 30  |
| 5.8 | Summary of the model developed in Keras                                | 30  |
| 6.1 | Plot of Training and Validation Loss                                   | 37  |
| 6.2 | Plot of Training and Validation Accuracy                               | 37  |
| 6.3 | Output produced by a Hi-En Transliteration Tool                        | 38  |

# List of Tables

|     |                                                                                                      |    |
|-----|------------------------------------------------------------------------------------------------------|----|
| 4.1 | Examples of Hi-En Code Mixed Comments from the dataset                                               | 15 |
| 4.2 | Illustration of free structure present in code mixed text. All sentence convey the same meaning      | 16 |
| 4.3 | Spelling variations of words in the Hi-En code-mix dataset                                           | 16 |
| 6.1 | Classification results depicting the performance of proposed system over sub-word and character LSTM | 38 |

# 1. Introduction

## 1.1 What is Code Mixing?

Code Mixing is a natural process in which linguistic units such as phrases, words, or morphemes of one language are used into an utterance of another language. It is the mixing of two or more languages in speech. For example (तुम कैसे हो? is usually written as “Tum Kaise ho?”). Code-mixing is a common phenomenon which widely observed in multilingual societies like India, which over 20 official languages and most popular of which are Hindi and English. Today with over 400 million population of India using the Internet daily, the usage of Hindi has been steadily increasing as a way to write comments, feedback on the Internet. Code-mixing is considered similar to the use of pidgins but former may occur within a multilingual setting where people speak more than one language. There are some works that define code-mixing as the process of placing various linguistic units from two or more grammatical systems into a single sentence and speech context. At a very younger age, children belonging to multilingual environments produce utterances that combine the elements of both the languages. Recent studies have shown that code-mixing is a demonstration of code-switching in socially appropriate ways. Code Mixing is used to explore the mental structures of its speakers and determine their underlying language understanding abilities.

## 1.2 Sentiment Analysis

Sentiment Analysis is the Natural Language Processing (NLP) task that deals with the detection and classification of sentiments present in the texts. In sentiment analysis, a basic task deals with identifying the presence or absence of any sentiment in the text, while other tasks aim at determining the polarity of the sentences by categorizing them into one of the *positive*, *negative*, and *neutral* classes. There is a huge demand in the market related to sentiment analysis tasks as it allows businesses to identify the sentiment of its customers towards products, brands, or services by analyzing their

online conversation, comments, and feedback. For example: “I really like the new design of your website!” → *Positive*, “I do not like this product...” → *Negative*, “I have my salon appointment today” → *Neutral*. Sentiment analysis performs contextual mining of data that can extract subject information and helps businesses to understand the social sentiments of their brand, products. In the times of Deep learning the ability of extracting sentiments from text has improved significantly. Advanced AI techniques can act as an effective tool for performing in-depth research and analysis of such tasks. The sentiment analysis task is also known as emotion AI as it is used to quantify the subject’s emotion towards certain things. Sentiment analysis can be used as a **voice of customers** over products, brands, and businesses that are mostly generated from reviews, survey forms, online and social media data, etc. A basic type of sentiment analysis is to determine the polarity of text while some advanced version of sentiment analysis is used to extract the particular emotional states such as “*happy*”, “*angry*” from the text.

The approaches for sentiment analysis has evolved over the last few decades, starting from naïve Bayes classifier to currently used Deep learning RNN models. The performance has significantly improved by using Recurrent Neural Networks as it is designed to handle temporal or time-series data. The future aspect of sentiment analysis is to perform real-time analysis of data *eg. of a cell phone, camera, etc.* and extract its particular sentiment value.

## 1.3 Literature Review

Even now the task of Sentiment analysis of Code-Mixed text is somewhat new and very little work has been done in it. But there exist some of the promising works that have significantly contributed to the development of high-performance systems to extract the sentiments of code mixed data. For now no such work has able to surpass even 70% accuracy mark and the major reason is because of existing of a lot of noise in the dataset. Several variations of the same words, the presence of multiple sentiments in a single text are some of the major data issues that lead to noise in the dataset. Some of the research papers that has been published and got significant improvement are listed below:

A paper named **Towards Sub-Word Level Compositions for Sentiment Analysis of Hindi-English Code Mixed Text** used the technique of sub-word level compositions to train the model. It used a convolution layer over the sequence of characters present in each training example and fed the output of the convolution layer to LSTM for training and prediction purposes.

Another research work is done by **Joshi et al. (2010)** in which the authors did the seminal work in sentiment analysis of Hindi text. The model was built on a three-step fallback model consisting of classification, machine translation, and sentiment lexicons. One of their key observations were that their system performed best with unigram features.

In a research work done by **Sharma et al. (2015)**, segregation of words into Hindi and English were performed. Then individual sentiment scores were lookup from respective sentiment dictionaries and combined to calculate the final sentiment value of the sentence. The sentiment score was used to classify whether the sentence belonged to which of the following classes *positive, negative, or neutral*.

A paper titled **Sentiment Identification in Code-Mixed Social Media Text** used the concepts of machine learning to automatically extracts sentiments from given text. The researchers used better pre-processing

techniques to remove noise present in the dataset and then used a multi-layer Perceptron model to determine the polarity of the sentiment. The pre-processing step is extensive and used techniques like Expansion of abbreviations, Removal of punctuations and Normalization of Words to remove any kind of noise that may be present in the dataset.

In another paper entitled **Automatic Normalization of Word Variation in Code-Mixed Social Media Text**, the authors tried to leverage the contextual property of words where the different spelling variations of words share a similar context in a large noisy social media data. The approach is to capture different variations of words that belong to the same context in an unsupervised manner. The developed model improves the performance in state-of-the-art Part-of-Speech (POS-tagging) & sentiment analysis tasks.

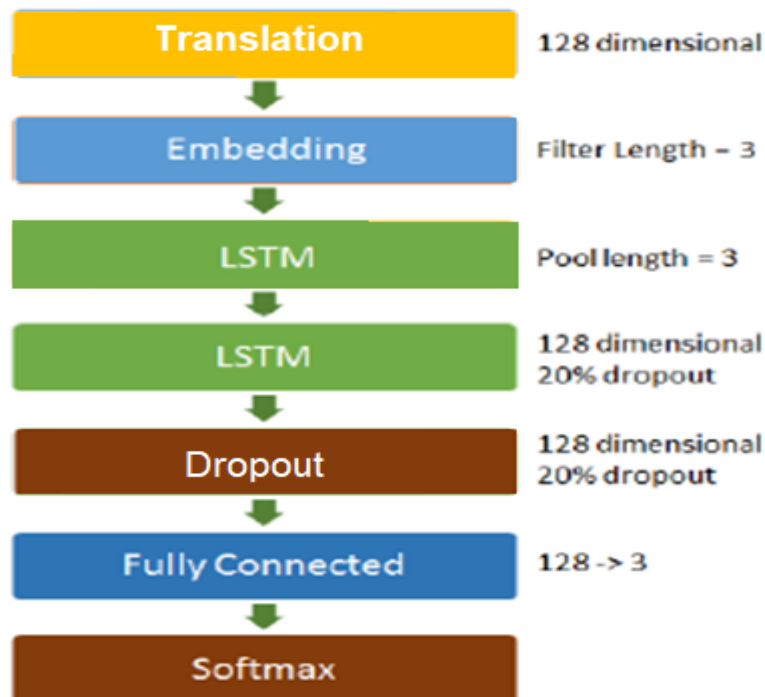
A thorough review of the above papers revealed that most of the paper focused on extracting the sentiment value of each sentence and used to determine the actual sentiment class of given sentence. All the existing techniques suffered from the noise present in the social media code-mixed dataset and that is why the performance of the existing system did not surpass even 70% mark.

A goal of this report is to develop a technique to automatically extract sentiments of given social media data handling the previously existing limitations and thus improve the performance of sentiment analysis tasks.



## 1.4 What I focused on

In this project, I try to automatically extract sentiments (*positive, negative or neutral*) from given code-mixed social media data using the concepts of Recurrent neural networks (RNNs) and used a multi-layered word-level LSTM model for training and developing the classification model to determine the sentiment of the text. The model is developed in Keras framework using Tensorflow as a backend. The words are preprocessed by translation API and then converted into embedding vector with the help of pre-trained GloVe word embeddings. The sequence of embedding vectors is fed to a multi-layered LSTM model for training and finally classified with the help of a softmax classifier. Also, the performance of the model is compared with a pre-existing architecture that uses a character-level model and sub-word level model for Sentiment analysis.



**Fig 1.1:** Proposed Model Architecture.

## 2. Problem Statement

### 2.1 Problem Definition

In Today's world as the use of the Internet grows day-by-day and the number of online users is increasing, there are a lot of textual data in the form of comments are collected online. Especially in multilingual societies like India, these data are mostly in the form of Code-Mixed text. These data can be analyzed to determine the sentiments of users towards any product, brands, or the company itself. It is therefore a need to be able to extract the sentiments of users and use it for business and other useful purposes.

The Problem Statement here specifically is to develop a Deep Learning model using Recurrent Neural Network that can be able to automatically extract sentiments (*positive, negative, or neutral*) from given code-mixed social media data. Also, while developing the architecture pre-process the dataset so that the noise is reduced and the performance of the model is as high as possible. Along the way, it is also required to explore and compare the performance of the developed model with that of the existing ones. Furthermore, try to scale the model so that it can be able to perform well on a larger dataset and reduce the variance problem so that it can be able to generalize well on new testing examples.

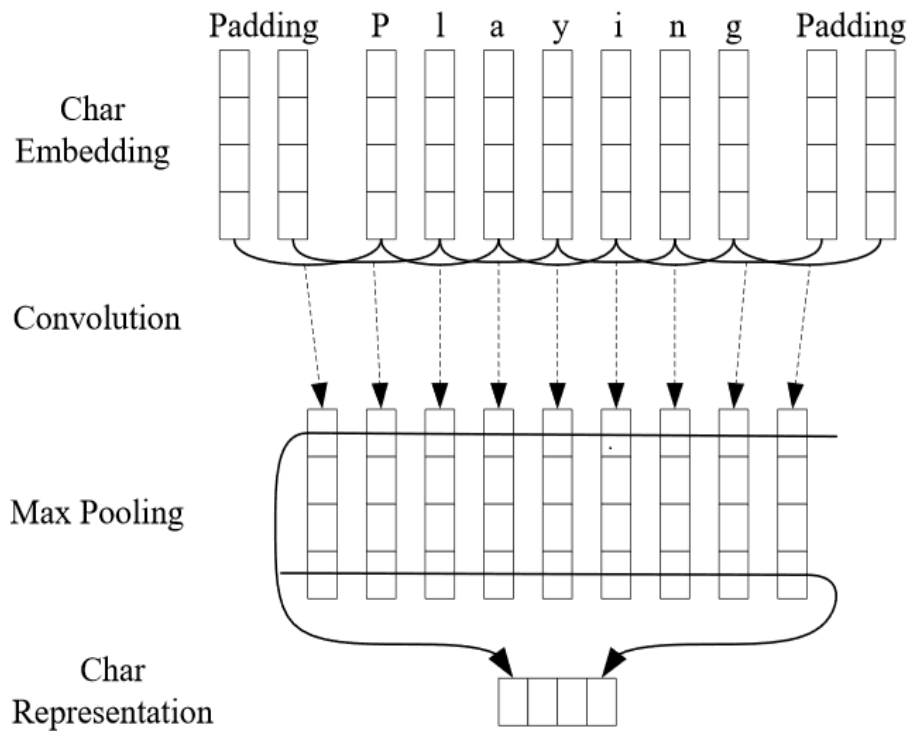
## 3. Existing Method & Limitations

### 3.1 Character-Level Models

#### 3.1.1 Introduction

The Character-level RNNs have recently become popular because of its coverage on a lot of problems related to NLP. In a character-level model, the input at each time step is an alphabet character or its respective embeddings. This allows us to minimize the vocabulary size of training and hence a lot of effort is reduced. One big advantage of using a character-level RNNs is that they can freely learn to generate new words. But this freedom, in fact, is an issue: As in any language words are the lexical units that are made by combining words in a specific manner, and a lot of combinations of characters would not have any meaning in the standard dictionary of words. *i.e.* most of the combinations of letters do not make any sense. The complexity arises because the mappings between the meaning of words and its construction from characters are arbitrary. Character models may be proper models of language as characters usually do not provide any semantic information as a whole, it is the specific combination of characters that make sense. For example, while “*King-Man+Women=Queen*” makes sense and is semantically interpretable by humans, but “*Cat-C+B=Bat*” does not make any proper sense and lacks any linguistic basis.

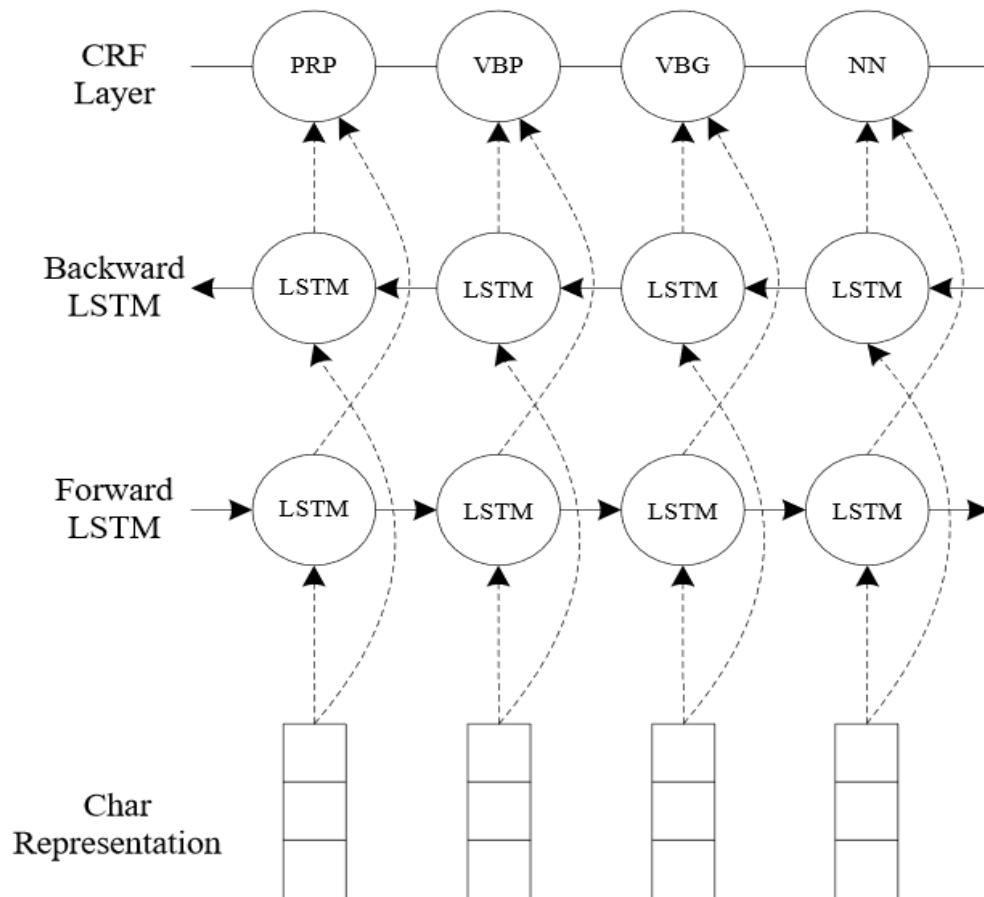
Some other disadvantages of using a character-level model is that number of parameters is very high and it requires a lot of training epochs to just train a simple model. Character-level models are used to solve some typical problems of Natural Language Processing where a fixed and definite vocabulary cannot be constructed from the available dataset. In such cases, character models can easily be implemented to map the input to output values as in a character-model there is no limitation of vocabulary.



**(a) CNN to obtain character-level representations. Dotted lines indicate dropout.**

**(b) BiLSTM network with CRF decoding layer. At the input, word embedding is concatenated with character representation.**

**Fig 3.1(a):** Typical Character-level model for Sentiment Analysis.



**Fig 3.1(b):** Typical Character-level model for Sentiment Analysis.

### 3.1.2 Limitations of Character-Level Model

1. Character models usually do not provide any semantic information as a whole. For example, “*King - Man + Women = Queen*” makes sense but “*Cat - C + B = Bat*”.
2. Number of parameters is very high and it requires a lot of training epochs to just train a simple model.
3. They have very small Embedding vectors (26 English alphabets), therefore don't have high accuracy.

## 3.2 Sub-Word Level Representations

### 3.2.1 Introduction

A Sub-word level representation can generate meaningful lexical representation. They can individually carry semantic weights. These sub-word representations are thought to believe that its composition of characters might allow the generation of new lexical structure that still preserve their meaning in linguistic units and even serve as a better model for Natural Language Processing (NLP) tasks. In a sub-word level representation, we use intermediate sub-word feature representations learned by the filters during convolution operation. In contrast to the traditional approaches that add the sentiment score of individual words, in a sub-word model, the relevant information is propagated with LSTM layers and then the final sentiment of the sentence is computed.

### 3.2.2 Hypothesis

It can be seen that incorporating sub-word level representations in the designs of the models can result into better performance of the model. This can also be used as a benchmark that can serve a test scenario for a broader hypothesis. The sub-word representations can incorporate linguistic prior to network architectures that can lead to better performance.

### 3.2.3 Methodology

To generate the sub-word representation most commonly used technique is using a 1-Dimensional Convolution on character input for a given sentence. In formal terms, suppose ' $\mathbf{C}$ ' be the set of all possible characters, and ' $\mathbf{T}$ ' be the set of all input comments/sentences. Each sentence is made up of words whose basic unit is the characters present in the character dictionary. Suppose a sentence  $\mathbf{S} \in \mathbf{T}$  is made up of a sequence of characters  $[c_1, c_2, \dots, c_l]$  where ' $l$ ' is the length of the input sentence ' $\mathbf{S}$ ' and  $c_i \in \mathbf{C}$ .

The input  $S$  can be represented by  $S \in \mathbf{R}^{d \times l}$  where ‘ $d$ ’ is the dimensionality of the character embeddings and ‘ $l$ ’ is the length of sentence  $S$ . To generate the sub-word a 1-Dimensional convolution  $Q$  needs to be performed on character embedding  $S$  with a filter size of  $H \in \mathbf{R}^{d \times m}$  where ‘ $m$ ’ is the length of the filter. After applying a convolution a bias is added and then a non-linearity activation function is applied to obtain a feature map  $f \in \mathbf{R}^{l-m+1}$ . The output of the convolution layer will produce the sub-word level (morpheme-like) feature map. The  $i^{\text{th}}$  element of feature  $f$  is given by

$$f[i] = g((Q[:, i:i+m-1] * H) + b)$$

where  $Q[:, i:i+m-1]$  is the matrix of  $(i)^{\text{th}}$  to  $(i+m-1)^{\text{th}}$  character embeddings and ‘ $g$ ’ is some non-linear activation function. Finally, the maximal responses are pooled out of a feature map  $f$  using a max-pooling layer ‘ $P$ ’ and the corresponding selected sub-word representations are given by

$$Y[i] = \max(f[P*(i:i+p-1)])$$

### 3.2.4 Model Architecture

To model the relationships from ‘ $Y$ ’ to its correct sentiment value, a multi-layered Long-Short Term Memory (LSTM) layer is used. As LSTM layers are well suited to learning to propagate and remember useful information, they can very well learn the mapping of temporal data features to their corresponding outputs. The feature vector ‘ $Y$ ’ is given as an input to the LSTM layer which performs some mathematical operations over a certain time step. I provided  $X_t$  input at ‘ $t$ ’ time step and  $A_{t-1}$  is the activation of the previous time step. The values of forget gate, update gate, and output gates are calculated along with the activations  $C_t$  and  $\hat{C}_t$ . The final output activation is calculated by element-wise multiplication of the output gate and the linear combination of activations calculated above. The final step’s output is passed through a fully connected layer which calculates the sentiment polarity.



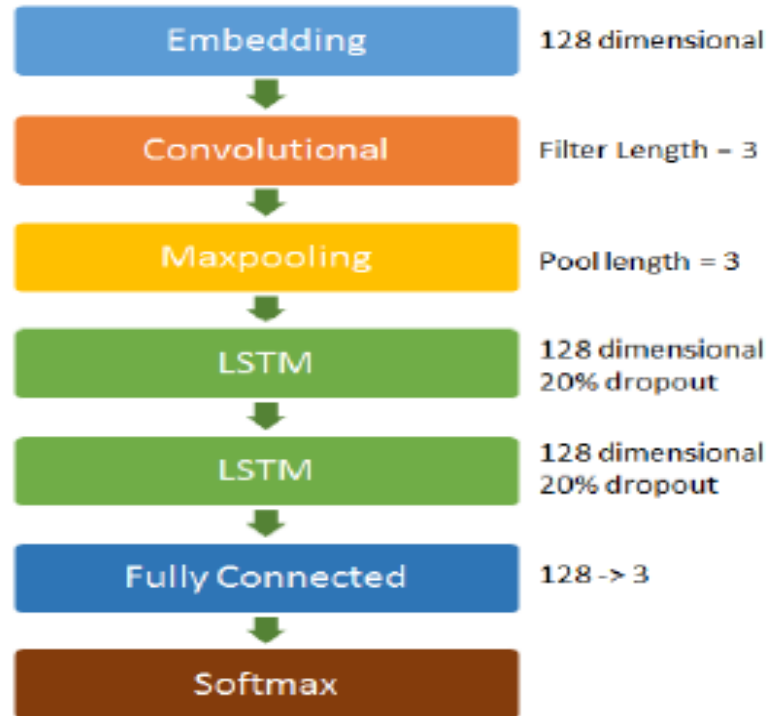
We can compute the output feature representation of LSTM using the following mathematical equations:

$$O_t = \sigma(WY + Uh(t - 1) + V(C_t + b))$$

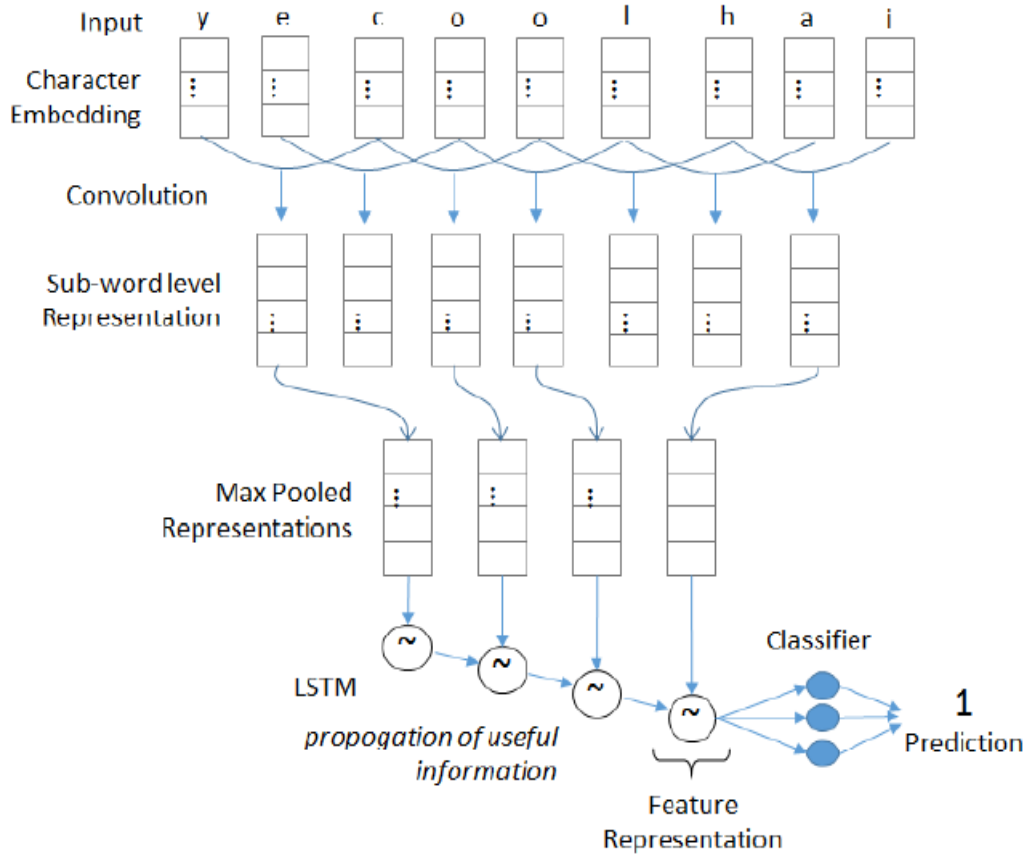
$$H_t = O_t * \tanh(C_t)$$

Where,

W, U, and V are the Weights matrices that form the learning parameters. The output H is then passed through a fully-connected dense layer which calculates the final sentiment polarity.



**Fig 3.2:** Schematic overview of the model Architecture.



**Fig 3.3:** Illustration of the proposed methodology.

### 3.2.5 Limitations of Sub-Word Level Model

1. A lot of combinations of characters would not have any proper meaning in any given language.
2. The non-meaningful sub-words generated after convolution can act as a noise value and degrade the overall performance of the model.

## 4. Dataset

The Dataset is collected from the user comments from public Facebook pages popular in India. It includes the pages of Salman Khan, a popular Indian actor with a massive fan following, and Shri. Narendra Modi, the current Prime Minister of India. These pages have 35 million and 40 million Facebook user likes respectively. These pages attract a large variety of users from all across India and contain a lot of comments on the original posts in code-mixed representations in varied sentiment polarities. From the set of comments, those longer than 50 words or containing only English words were removed. The comments were annotated in a 3-level polarity scale – *positive, negative, or neutral*. The dataset contains 15% negative, 50% neutral and 35% positive comments owing to the nature of conversations in the selected pages.

| Example               | Meaning            | Sentiment |
|-----------------------|--------------------|-----------|
| Salman khan pagal hai | Salman Khan is Mad | Negative  |
| Hello Bhai kaise ho?  | How are you, Bhai? | Neutral   |
| Mera PM Mahan.        | My PM is great.    | Positive  |

**Table 4.1:** Examples of Hi-En Code Mixed Comments from the dataset.

### 4.1 Some Major Data Issues

The Dataset exhibits some of the major issues while dealing with the code-mixed data like:

- Short sentences with unclear grammatical structure.
- Number of variations of how words can be written.
- Presence of multiple sentiments in a single comment.
- Multiple variations of comments conveying the same sentiment.

| Sentence variations    |
|------------------------|
| Movie dhanmsu hai bhai |
| Dhanmsu film hai bhai  |
| Bhai film maal hai     |
| Bhai maal picture hai  |

**Table 4.2:** Illustration of free structure present in code mixed text. All sentences do convey the same meaning.

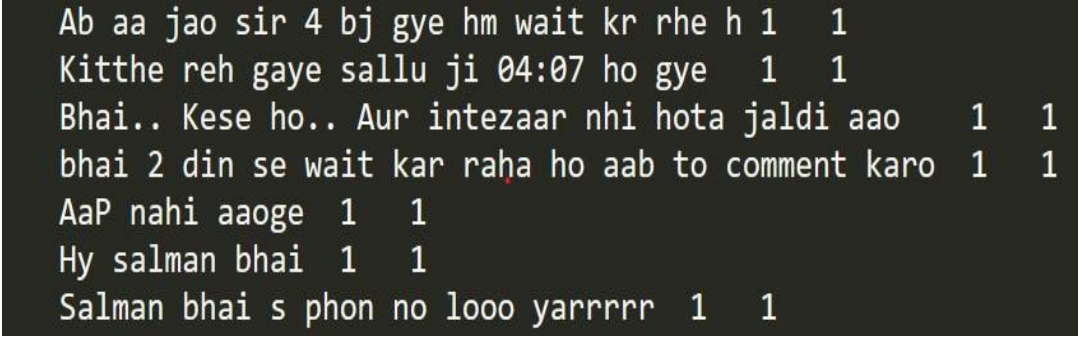
| Word                 | Meaning | Appearing Variations                                         |
|----------------------|---------|--------------------------------------------------------------|
| बहुत<br>(bahut)      | Very    | bahout bohut bhout<br>bauhat bohot bahut bhaut<br>bahot bhot |
| प्यार<br>(pyaar)     | Love    | pyaar peyar pyara piyar<br>pyr piyaar pyar                   |
| मुबारक<br>(mubaarak) | Wishes  | mobarak mubarak<br>mubark                                    |

**Table 4.3:** Spelling variations of words in code-mixed dataset.

## 4.2 Data Preprocessing

Data preprocessing is one of the most important steps in developing any Machine Learning or Deep Learning model. It is a data mining technique that is used to transform the raw data into a more useful and efficient format. In other words, data is transformed to bring it to such a state so that the machine can easily parse it. In other words the features of the data can now be easily be interpreted by the algorithm. During pre-processing noise, outliers and incorrect data are removed from the dataset. Any kind of non-mathematical data is converted into mathematical form by using its respective features. This can either be done using one-hot encodings or Embedding vectors.

The extracted data from the Facebook pages is in the text format. It is loaded into the python code and split into separate lines using python's **split()** function. Each line of comment is used then parsed and its annotated sentiment value is collected. Also, each word in the comment is translated into a common language (English in this case) using **Google Translation API (googletrans)**. Cloud translation is a feature provided by Google that can dynamically translate text between thousands of language pairs. This translation enables websites and programmers to integrate with their customers or products.



```
Ab aa jao sir 4 bj gye hm wait kr rhe h 1 1
Kitthe reh gaye sallu ji 04:07 ho gye 1 1
Bhai.. Kese ho.. Aur intezaar nhi hota jaldi aao 1 1
bhai 2 din se wait kar raha ho aab to comment karo 1 1
AaP nahi aaoge 1 1
Hy salman bhai 1 1
Salman bhai s phon no looo yarrrrr 1 1
```

**Fig 4.1:** Comments collected from the Facebook page (Before Translation).

```
1 from googletrans import Translator
2 translator = Translator()
3 word = 'khushi'
4 translated = translator.translate(text = word, dest = 'en', src = 'auto')
5 print(translated.text)
```

☞ Happy

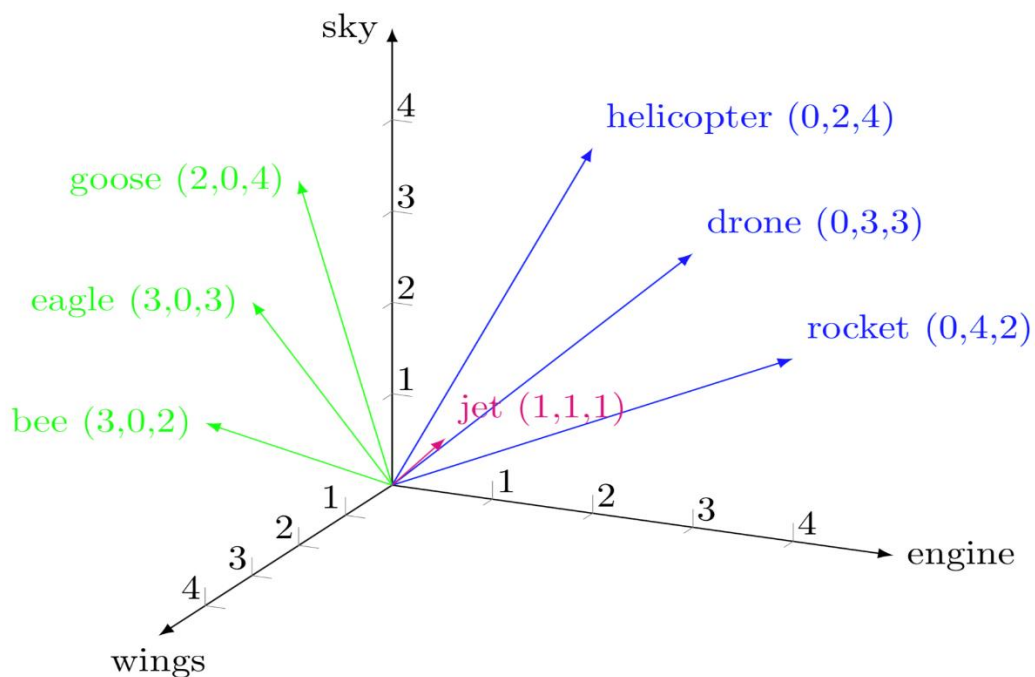
**Fig 4.2:** Example script of Google Translation API.

```
37 ['we', 'are', 'proud', 'our', 'pm', 'from', 'lg', 'stayed', 'of',
   'ass', 'country', 'one', 'kcci', 'rah', 'per', 'movable', 'for',
   'is', 'oh', 'well', 'i', 'know', 'hm', 'puri', 'world', 'me',
   'ours', 'flag', 'special', 'can', 'is']
38
39 ['modi', 'show', 'up', 'you', 'of', 'hair', 'very', 'actually',
   'seems', 'is', 'progress', 'here', 'hi', 'all', 'face', 'is',
   'is', 'for', 'from', 'all', 'necessary', 'is', 'of', 'country',
   'or', 'vikas', 'guarantee']
```

**Fig 4.3:** Comments generated using Google Translation API (After Translation).

### 4.3 Words To Embeddings

A word embedding is a learned representation of text where words are represented in the form of higher dimensional vectors. It is one of the most important data preprocessing step that allows data to be converted into its respective Embeddings. Word level models are the most commonly used statistical approaches to learn word-level feature representations. **Word2Vec** (Mikolov *et al.* 2013) and **Word-RNNs** (thang Luong *et al.* 2013) have substantially contributed to the development of vector representations of English words and their applications in various NLP tasks such as Summarization, Machine Translation and Sentiment Analysis. They are theoretically sound since language consists of inherently arbitrary mappings between words of different languages. One of the most common approaches in using the Word2Vec model is to use a pre-trained Embedding that has already been trained on a very large corpus of data. One such Embedding is the GloVe embedding provided online by Stanford University.



**Fig 4.4:** Vector representation of a word using Word Embeddings.

### 4.3.1 GloVe Embeddings

*GloVe: Global Vectors for Word Representation.*

#### Introduction

The co-occurrence of words in a corpus teaches us about its meaning. It is impossible to define the characteristics or properties of each word manually. Word Embedding is a Deep learning method in deriving vector representation of words. There are many techniques to automatically learn word embeddings. One of the ways is to use the Skip-Gram model or Global vector method. In this project Glove embeddings are used. The Glove is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated word-to-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructure on the word vector space. To understand GloVe, the following terms are defined:

$X_{ij}$  tabulate the number of times word  $j$  occurs in the context of word  $i$ .

$$X_i = \sum_k X_{ik}$$

$$P_{ij} = P(j|i) = X_{ij}/X_i$$

The Ratio of co-occurrence of probabilities is given by:

The diagram illustrates the GloVe co-occurrence ratio formula. At the top left, the text " $w \in \mathbb{R}^d$  are word vectors" has arrows pointing to the vectors  $w_i$  and  $w_j$  in the formula. At the top right, the text "probe word" has an arrow pointing to  $\tilde{w}_k$ . The formula itself is  $F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$ . Below the formula, the text "co-relations between the word  $w_i$  and  $w_j$ " has arrows pointing to  $w_i$  and  $w_j$ . The text "co-occurrence probabilities for the word  $w_j$  and  $w_k$ " has an arrow pointing to the denominator  $P_{jk}$ .

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

This ratio indicates some correlation of word  $W_k$  with words  $W_i$  and  $W_j$ . This correlation is also used to determine the cost function of training. The cost function tries to maximize the similarity between correlated words and as a result of training over a huge corpus of data a well-suited vectors representation of words is obtained.



| Probability and Ratio | $k = solid$          | $k = gas$            | $k = water$          | $k = fashion$        |
|-----------------------|----------------------|----------------------|----------------------|----------------------|
| $P(k ice)$            | $1.9 \times 10^{-4}$ | $6.6 \times 10^{-5}$ | $3.0 \times 10^{-3}$ | $1.7 \times 10^{-5}$ |
| $P(k steam)$          | $2.2 \times 10^{-5}$ | $7.8 \times 10^{-4}$ | $2.2 \times 10^{-3}$ | $1.8 \times 10^{-5}$ |
| $P(k ice)/P(k steam)$ | 8.9                  | $8.5 \times 10^{-2}$ | 1.36                 | 0.96                 |

Very small or large:  
 solid is related to ice but not steam, or  
 gas is related to steam but not ice

close to 1:  
 water is highly related to ice and steam, or  
 fashion is not related to ice or steam.

The cost function of training is given by:

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

$$f(x) = \begin{cases} (x/x_{\max})^{100} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$

## Key Features of GloVe:

- [1] **Nearest Neighbours:** The cosine similarity between words pertaining to the similar meaning is very high and those words are in very close proximity to each other. According to metric the nearest neighbors are rare and such relevant words that are far beyond the normal human vocabulary.

*For example: Frog  $\rightarrow$  Toad, Leptodactylidae.*

- [2] **Linear Substructure:** The similarity metric used for nearest neighbor evaluation produces a single scalar that quantifies the relatedness of words. Man can be kept similar to the woman on some axis, these words highlight a primary axis along which the human differs from each other.

*For example: Man: Woman::King: Queen.*

## Embeddings Overview

In this project, I have used *glove.6B.100d.txt* file that contains word embedding that is trained on a corpus of 6 Billion words where each embedding is a 100-Dimensional vector. The words and vectors are stored in the form of a dictionary mapping which can be easily loaded in a python file. These embeddings are trained by researchers at Stanford University NLP team and are provided online for use over any task related to Natural Language Processing.

## Training of GloVe Vectors

The GloVe model is trained based on the non-zero entries of a global word-to-word co-occurrence matrix, which can tabulate how frequently words can co-occur with one another in the given corpus of text. To populate this matrix it requires a single pass through the entire corpus and collects the statistics. For large corpora, this type of pass can be very computationally expensive, but this cost is a one-time consumable cost, as once trained the embeddings can be used along with any problem related to the NLP domain. Further training iterations are much faster as the number of non-zero matrix entries is typically much smaller than the total number of words in the corpus. There are tools provided in this package that can automate the collection and preparation of co-occurrence statistics for input into the model. The core training code is a separate one from these preprocessing steps and can be executed independently whenever required.

```
[15] 1 print(embeddings_dict['engineer'])
```

```

↳ [-0.22299 -0.50327 -0.1059 -0.12002 0.5725 -1.3863
    0.76886 -0.57806 -0.42936 1.1056 0.19968 -0.4566
    0.50304 -0.17551 -0.098026 -0.69218 0.81451 -0.36316
    -0.61163 0.24944 -0.35084 0.033094 0.74037 -0.73754
    -0.42547 -0.66803 -0.42648 -0.1754 0.69981 0.14086
    -1.1764 0.85203 -0.63575 -0.17872 -0.3841 -0.2632
    -0.49453 0.0015678 1.4436 0.041463 -0.1215 0.1022
    0.22932 -0.40678 0.45662 0.24618 -0.45261 -0.31579
    -0.15661 -0.18254 -0.43065 -0.33737 -0.079198 0.92812
    0.35323 -1.263 -0.82774 -0.0054649 0.94668 0.23276
    0.87403 0.16087 0.19137 0.73792 0.15513 -0.21225
    0.3676 0.4208 -0.012074 0.86419 -0.23969 0.63741
    -0.31752 -0.34015 0.31799 -0.018631 1.0928 0.27653
    -0.054411 -0.44682 0.34423 0.075848 -0.22979 0.65667
    -1.0455 0.30537 0.51792 -0.71699 -0.29943 -0.73867
    0.61258 -0.63905 0.048985 0.3719 0.096472 0.32427
    -0.096779 -0.79171 -0.52218 -0.84882 ]

```

**Fig 4.5:** An Example 100-Dimensional embedding vector of GloVe Word Embedding.

## 4.4 Using Glove Embeddings

After the translation of sentences to English, the translated data/comments are loaded along with the pre-trained GloVe embeddings. For each word present in the translated comment, it's corresponding word embedding is checked. If it exists then the corresponding embedding is appended to the embeddings\_vector, otherwise, a 100-Dimensional vector of all zeros is appended to signify the absence of the word in the GloVe embeddings. Each training example is padded up to 50 embeddings while each word embedding is a 100-Dimensional vector.

```
[ ] 1 embeddings = create_embedding_vectors(Masterdir,translated_data)
    2 print(embeddings[0][MAXLEN-1])
```

```
mx : 110
(2307, 50, 100)
[-0.17763291 -0.07153087  1.02335025 -0.32432374  0.39428754 -0.37721835
 1.48816441  0.74926245  0.26568073 -0.42017986  1.57657678 -0.33609742
-1.36831988  0.37420312  1.19760731 -2.03527542 -0.13874713  1.092029
 0.42859453 -0.89155821 -0.04424921 -0.8746607  -0.72301644 -0.8374238
-0.61565568  0.6046272  0.51119385  0.90035755  0.69706549  0.33644496
 0.23554043 -0.46773462 -1.90049781 -0.27068867  1.58446773 -0.9847348
-0.33490869  2.61477296 -2.14947244 -0.37663543 -0.56198022  0.72809516
 2.32685344  0.73665284  1.67240344 -0.2789665  0.57326702 -2.05041681
 1.46615088 -0.84031951 -0.41184002  0.36121305  0.03813054  1.86335749
-0.82500947  1.31746601  1.17497619  1.72591234  0.05347791  0.03446738
-1.33192183 -0.29780169  1.29515553  2.84886365  1.55303261  0.57199673
 1.16561945  0.10756005  1.09983155 -0.09807004 -2.25945202 -0.70120466
-0.39054531  0.11729553  2.02580477 -1.12014516 -0.23888191 -0.50850811
 0.88612207  0.63229186 -0.1564924  1.23942472  0.16858559  1.1389902
 0.99971771  0.62518252  0.37401078  0.89942653  0.82382083  0.24945688
-1.51671418  1.16410371 -1.70020692  1.62070849 -0.07543785 -1.28626272
-0.555787  0.53581096  1.03735178  0.08536558]
```

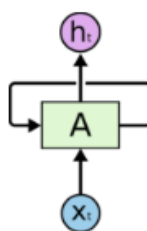
**Fig 4.6:** Creation of Embedding vector from translated words.

# 5. Model Overview

## 5.1 Introduction

Recurrent Neural Networks (RNNs) are one of the powerful tools to develop deep learning models for analyzing time-series data. In RNNs the connection between nodes create a directed graph along the temporal sequence which allows it to exhibit temporal dynamic property. Each RNN cell has some internal memory to process variable-length inputs. The basic principle behind the working of RNNs is that the previously computed outputs/activations along with the input of the current time step is used to derive the current time step output/activation.

RNN finds application in various fields related to Natural Language Processing like Machine translation, Speech recognition, Sentiment Analysis, Language model, etc.



**Recurrent Neural Networks have loops.**

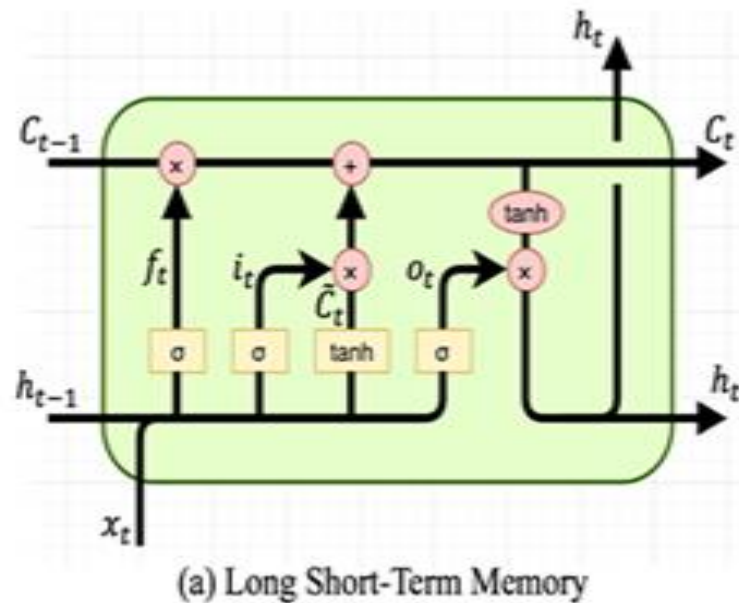
**Fig 5.1:** Representation of a time sequence RNN model.

### 5.1.1 Basic RNN Learning Cell

- **Long-Short Term Memory (LSTM)**

They were invented by Hochreiter and Schmidhuber in 1997 and set accuracy records across multiple application domains. Models trained on LSTM has outperformed their own benchmarks and brought a revolution to the language processing methods.

LSTM has broken the records for Language modeling, Machine translation, and Automated Image captioning.



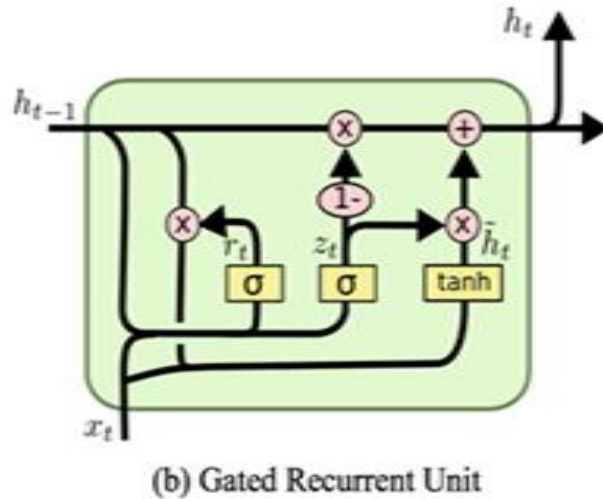
**Fig 5.2:** General architecture of an LSTM cell.

$$\begin{aligned}
\tilde{c}^{<t>} &= \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \\
\Gamma_u &= \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \\
\Gamma_f &= \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \\
\Gamma_o &= \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) \\
c^{<t>} &= \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} \\
a^{<t>} &= \Gamma_o * \tanh c^{<t>}
\end{aligned}$$

**Fig 5.3:** Mathematical computation of a single time step in the LSTM cell.

- **Gated Recurrent Units (GRU)**

GRUs were introduced in 2014 by *Kyunghyun Cho et al.* It has a similar working principle to that of an LSTM cell but with fewer parameters. Due to its simplicity, it has become one of the favorable choices of building an RNN model. GRUs even tend to show better performance on certain smaller datasets.



**Fig 5.4:** General architecture of a GRU cell.

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

**Fig 5.5:** Mathematical computation at a single time step in the GRU cell.

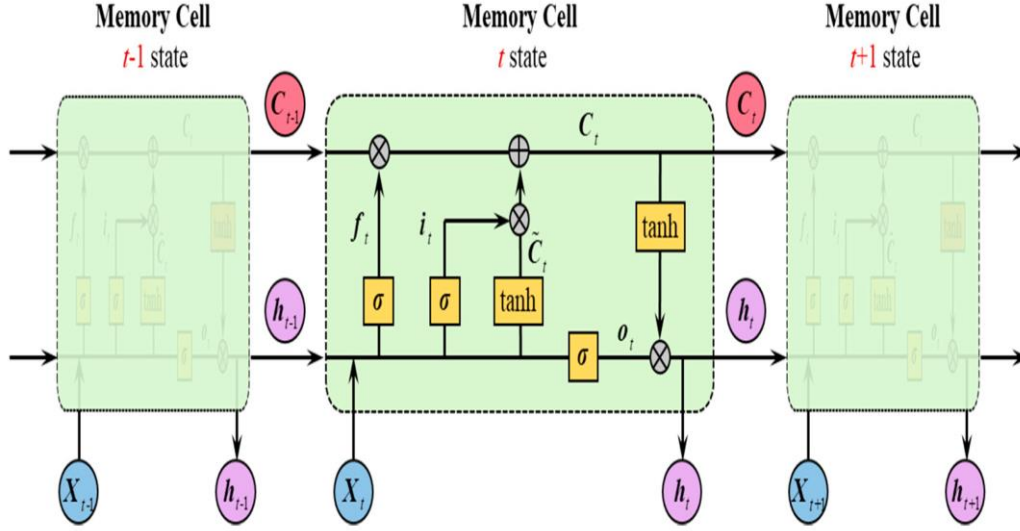
## 5.2 Objective

To develop a multi-layer LSTM model that takes the embedding vectors as input at each time step (There are a total of 50 time steps). Perform linear combination and pass the result through some activation function and after doing all mathematical calculations predict the probability of sentence/comment belonging to different sentiment classes ( $0 \rightarrow Negative$ ,  $1 \rightarrow Neutral$ ,  $2 \rightarrow Positive$ ).

### 5.2.1 Methodology

The representation of one input training example is given by the matrix  $Q \in \mathbf{R}^{d \times l}$  where ‘ $\mathbf{R}$ ’ is a set of Real numbers, ‘ $\mathbf{d}$ ’ is the dimension of one embedding vector and ‘ $\mathbf{l}$ ’ is the total time steps. In the project, I used  $\mathbf{d}=100$  and  $\mathbf{l}=50$ . This data is used to map the relationship between features  $\mathbf{R}^{d \times l}$  and the sentiment output. The mapping is learned using a multi-layer LSTM model which is best suited in learning to propagate and remember useful information even after longer time steps, and finally arriving at a sentiment value prediction.

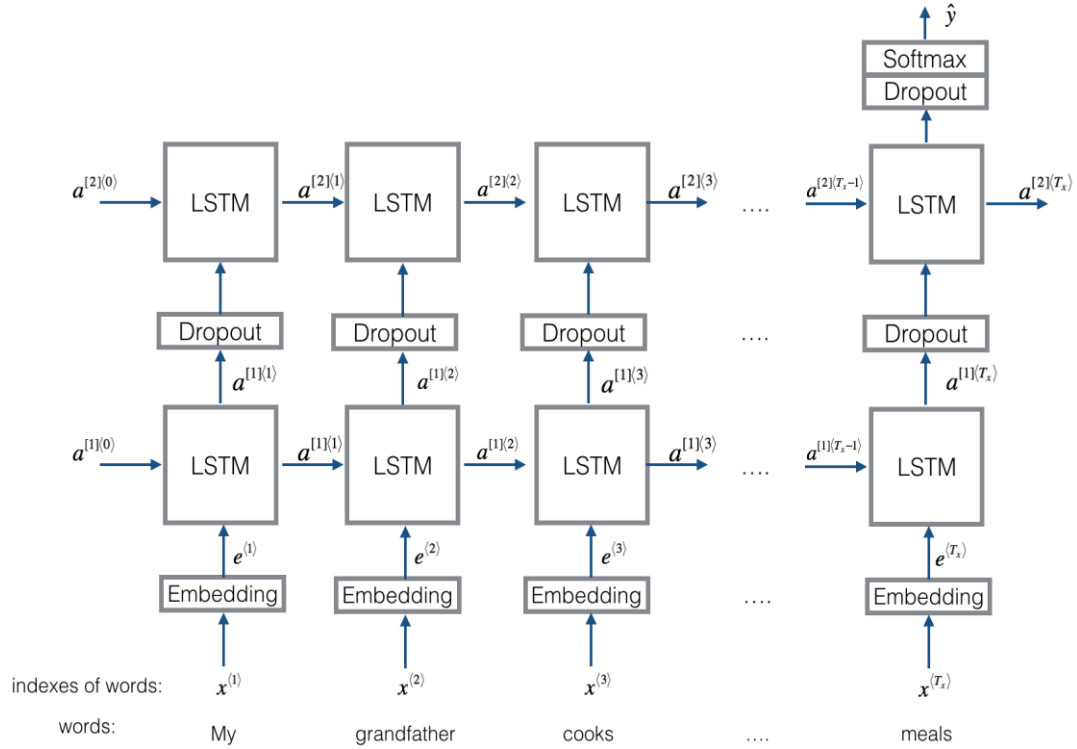




**Fig 5.6:** Instance of an LSTM cell at time step ‘t’.

I provided  $X_t$  input at ‘t’ time step and  $A_{t-1}$  is the activation of the previous time step. The values of forget gate, update gate, and output gates are calculated along with the activations  $C_t$  and  $\hat{C}_t$ . The final output activation is calculated by element-wise multiplication of output gate and the linear combination of activations calculated above. The final step’s output is passed through a fully connected dense layer which calculates the sentiment polarity.

## 5.3 Architecture



**Fig 5.7:** Illustration of the proposed methodology.

```
[24] 1 model.summary()
```

Model: "sequential\_1"

| Layer (type)              | Output Shape    | Param # |
|---------------------------|-----------------|---------|
| lstm_1 (LSTM)             | (None, 50, 128) | 117248  |
| lstm_2 (LSTM)             | (None, 128)     | 131584  |
| dense_1 (Dense)           | (None, 3)       | 387     |
| activation_1 (Activation) | (None, 3)       | 0       |
| Total params: 249,219     |                 |         |
| Trainable params: 249,219 |                 |         |
| Non-trainable params: 0   |                 |         |

**Fig 5.8:** Summary of the model developed in Keras.

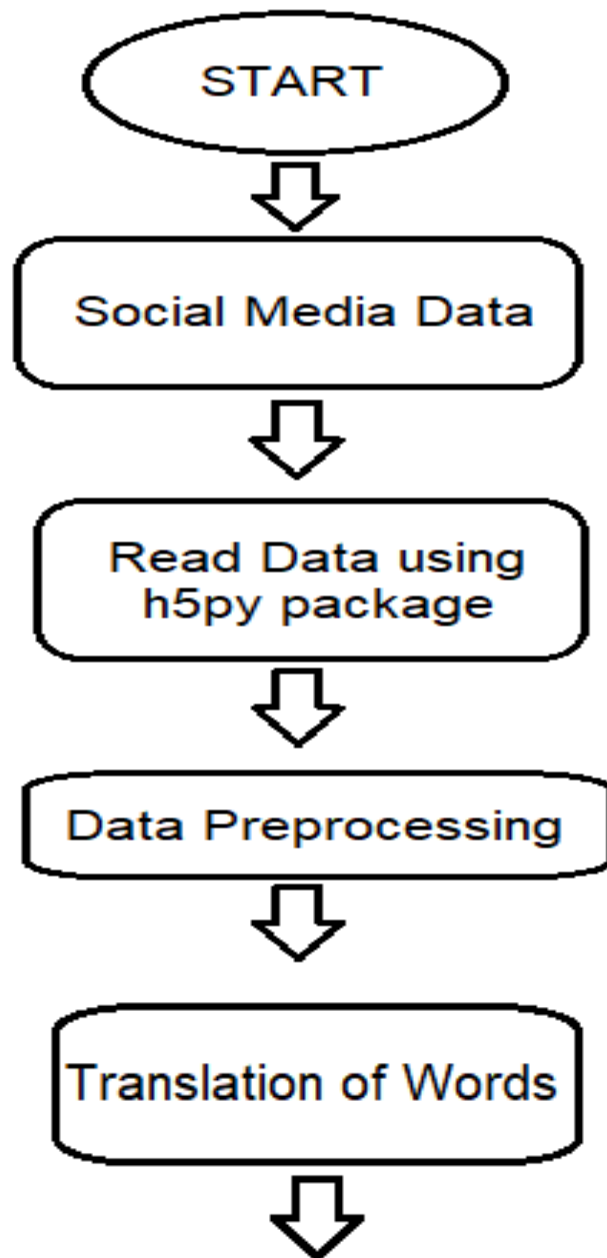
## 5.4 Experimental Setup

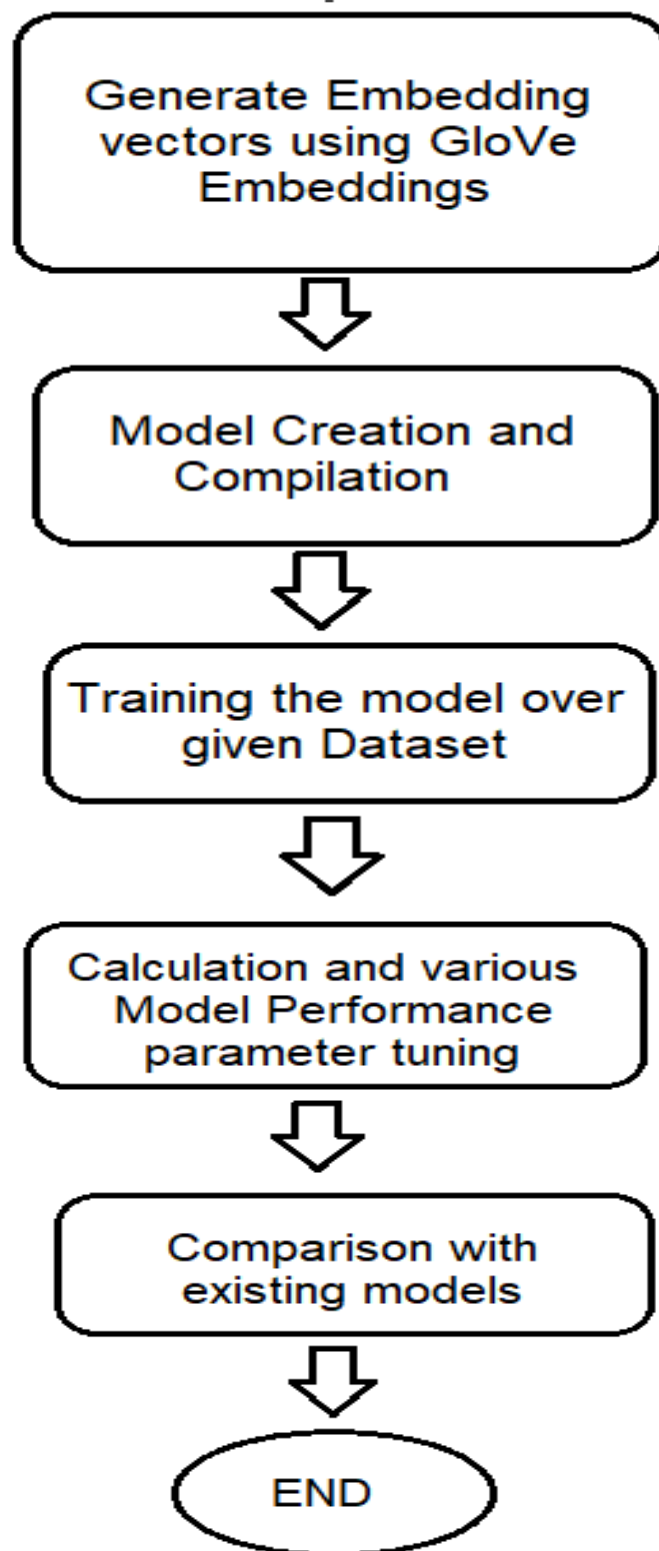
The dataset is divided into 3 splits- Training, validation, and testing. I first divided the data randomly into 80-20% train-test split, then further randomly divided the training data into 80-20% train-validation split to obtain the appropriate training, validation, and testing examples.

The model is developed in Keras framework using Tensorflow in the backend. The model contains a multi-layered LSTM layer to better learn the mappings between embeddings and sentiment classes. The output produced by the LSTM layers is passed through a fully connected Dense layer and finally, a Softmax classifier is used to predict the probability of data example belonging to different sentiment classes. The class having the maximum probability is considered as the sentiment for the training example under consideration.

The model uses a Categorical cross-entropy loss model and Accuracy as the performance metric. In order to improve the performance of training, Adam optimizer is used for smooth gradient descent. During training, Dropout is also applied to remove any kind of overfitting occurring in the system and to improve the test set accuracy. The validation data is also supplied to the model so that it can generalize over newer test examples and thus improve its performance. Training is done in mini-batches to optimize the backpropagation step.

## 5.5 Experimental Flow Chart





A brief description of various steps involved are:

- [1] **Social Media Data:** The social media data is collected from the comments present on Facebook pages of Mr. Salman Khan and Shri Narendra Modi and labeled in one of the following classes, *positive*, *negative* or *neutral*.
- [2] **Reading Data:** The raw data is loaded into the Machine Learning model using h5py package. Once loaded the data can be used for the pre-preprocessing step.
- [3] **Data Preprocessing:** It is one of the most important steps in building any machine learning application. In this step, noisy, outlier or missing data is handled so that the overall performance of the system can be improved.
- [4] **Translation of Words:** All the words are translated into a common language (English in this case) using a Translation API. The Translation API that was used in the project is googletrans. It is an API that is developed by Google and can be used by a paid account for translation between more than 300 language pairs.
- [5] **Generate Embedding Vectors using GloVe Embeddings:** In this step the pre-trained GloVe embeddings are used to convert each word into its respective embedding vector. Each embedding vector is of 100 Dimension and best describes the respective properties of words.
- [6] **Model Creation and Compilation:** A multilayer Recurrent Neural Network is created using LSTM as the basic learning cell to best learn the mapping from input embeddings to output sentiments. The model is developed in Keras framework and compiled using Categorical cross-entropy loss model with Adam optimizer.
- [7] **Training the Model:** The developed model is trained over many epochs to learn the input-output mapping. It performs forward

propagation step to calculate the cost and then propagates backward to calculate the gradient and update the parameters.

[8] **Calculation and Hyper-parameter Tuning:** The hyperparameters are tuned in such a way that the model achieves maximum accuracy and be able to generalize well even on newer testing examples.

[9] **Comparison with Existing Results:** The developed model is compared with the existing models to check the limitations of the developed model and tabulate the results obtained.

## 5.6 Method Suitability

There are some techniques that are used to perform sentiment analysis in only English based text:

- **NLP Tool based approach:** It involves the generation of parse trees which may not be available for the code-mixed dataset.
- **Approach based on Surface feature engineering:** Hashtags, User mentions, Emoticons, etc. should not be present in the data, which is simply not the case of code-mixed data.

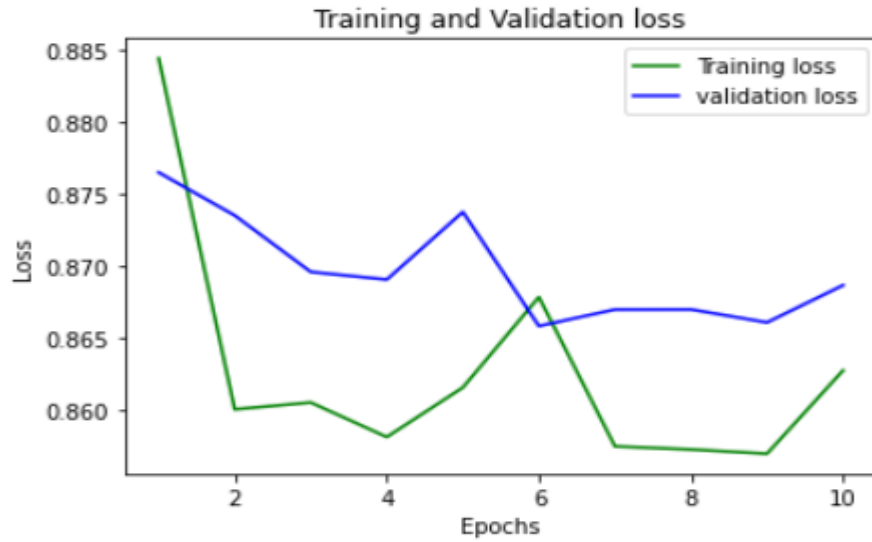
## 6. Observation

The model is developed in Keras framework was trained using more than 2000 training examples and tested on nearly 300 examples. There were certain important observations made during the training and testing phase of the model. These observations are listed below:

- [1] Due to the presence of a lot of misspelled words the model training accuracy is not surpassing a certain threshold.
- [2] The dataset includes multiple variations of the same word, which makes their translation to English a tough task even for highly efficient and trained Google Translation API.
- [3] There are certain words that are improperly written and makes no sense even in their native language, this makes training very tough and fewer chances improve accuracy.
- [4] The model's training accuracy can be improved by using more parameters for training or training the model for a large number of epochs.
- [5] On increasing the number of parameters of the model or number of epochs, the model starts to overfit the training set data, due to which the testing accuracy decreases.
- [6] To reduce the overfitting of data, dropout and recurrent dropout are used on dense and recurrent layers respectively. Using dropout the Variance of the model is reduced and the model is able to generalize to new examples present in the test set.

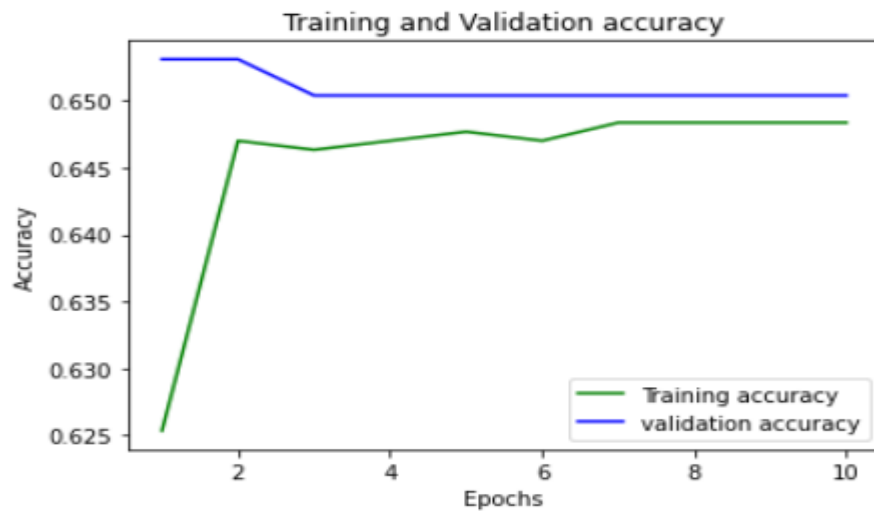


## 6.1 Plot for Training/Validation Loss



**Fig 6.1:** Plot of Training and Validation Loss.

## 6.2 Plot for Training/Validation Accuracy



**Fig 6.2:** Plot of Training and Validation Accuracy.

## 6.3 Comparing Results

I have used exactly the same dataset for the comparative study between the sub-word level model and the model proposed by me. The pre-trained weights of the sub-word LSTM model are loaded from .h5 file. An evaluation function was defined to determine the performance of the model on the given test data based on the Accuracy as a metric. The results produced are as follows:

| Model/Type                | Train Set Performance |          | Test Set Performance |          |
|---------------------------|-----------------------|----------|----------------------|----------|
|                           | Accuracy              | F1 Score | Accuracy             | F1 Score |
| <b>Char-LSTM</b>          | 59.8%                 | 0.511    | 46.6%                | 0.332    |
| <b>Sub-Word LSTM</b>      | 69.7%                 | 0.658    | 67.3%                | 0.637    |
| <b>Translation + LSTM</b> | 67.3%                 | 0.642    | 66.2%                | 0.652    |

**Table 6.1:** Classification results depicting the performance of proposed system over sub-word and character LSTM.

| Comment/Meaning                                                                                                                                                                    | Transliterated Comment                                                 | Observation                                                                                             |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|
| Bhai ied mubaraq<br><i>Wishes for Eid, brother</i>                                                                                                                                 | भाई आईद मुबरक                                                          | The words ied and mubaraq are spelt differently from usual form which caused incorrect transliteration. |
| Aatma aur mun ki pavitrata ki jhalak hai is beti<br>ki aawaz mei, khush raho beti<br><i>There's a glimpse of piouness of soul and mind in this girl's voice, stay happy, girl!</i> | आत्मा अर मुन की पवितराता की झालक है इस बेती की आवाज़ में, खुश रहो बेती | The words aur, pavitrata and beti are written as expected, but still incorrectly transliterated.        |
| love u sir love u soo much l'ts beautiful vedio<br><i>Love you so much sir. It's a beautiful video.</i>                                                                            | लव उ sir लव उ सू much उस ईट<br>beautiful vedio                         | love should have been identified as an English word, which expresses sentiment of the sentence.         |

**Fig 6.3:** Output produced by a Hi-En Transliteration Tool.

## **6.4 Validation of Proposed Hypothesis**

The obtained preliminary results for my hypothesis incorporating Google Translation API (googletrans) and GloVe Embeddings instead of sub-words would lead to similar performance. My proposed model has comparable accuracy of 66.2% on the test set which is very close to the accuracy of the sub-word LSTM model (67.3%). However, the F1-score of my model is much better than later. Furthermore, the overall performance of my model is much better than the character-level LSTM in both terms i.e. Accuracy and F1-score. In all the cases, the sentences were converted to lowercase format and then tokenized. Also, No extra heuristic or features were added or used.

## 7. Conclusion

I have proposed a new architecture for the Sentiment Analysis of a noisy Code-Mixed (Hi-En) Dataset. I discussed some of the issues present in such a code-mixed dataset due to which there is an unavailability of NLP tools for Sentiment Analysis. The solution that I proposed involved translation of the given dataset in the English language with the help of a Translation API. After the translation, the words are converted into embedding vectors using the pre-trained GloVe embeddings provided by Stanford University. These embeddings comprise a feature matrix, that is used over a multi-layer LSTM model to learn the mapping from input sentences to output sentiments. Furthermore, the performance of the proposed model was compared with that of character-level and sub-word level models, and the results depicted that the performance of the proposed architecture is comparable and sometimes better in comparison to the later.

### 7.1 Future Work

Future work would be to explore ways to remove noise from the dataset to improve the test set performance, and also to observe the effects of scaling of the proposed model with a large dataset and deeper RNNs.

# References

- [1] Basics of using pre-trained GloVe Embeddings in python.  
<https://medium.com/analytics-vidhya/basics-of-using-pre-trained-glove-vectors-in-python-d38905f356db>
- [2] Documentation for Google Cloud Translation API.  
<https://cloud.google.com/translate/docs>
- [3] Pre-trained GloVe Embeddings. <https://nlp.stanford.edu/projects/glove/>
- [4] Build a Machine Learning Model using Google Colaboratory.  
<https://colab.research.google.com/>
- [5] Keras Documentation. <https://keras.io/>
- [6] Preprocessing of Dataset using Kaggle GPU/TPU. <https://www.kaggle.com/>
- [7] Data Visualization in Deep Learning using Matplotlib.  
<https://www.pluralsight.com/guides/data-visualization-deep-learning-model-using-matplotlib>

# Appendix

## Code for Sentiment Analysis of Code-Mixed Text

```
[ ] 1 import os
    2 os.chdir('drive/My Drive/Dataset/CodeMixed')
    3 !ls
```

```
[ ] 1 pip install googletrans
```

```
▶ 1 import numpy as np
   2 import h5py
   3 import pickle
   4 from copy import deepcopy
   5 from sklearn.metrics import confusion_matrix
   6 from sklearn.model_selection import train_test_split
   7 from keras.models import Sequential, load_model
   8 from keras.callbacks import Callback, ModelCheckpoint
   9 from keras.wrappers.scikit_learn import KerasClassifier
  10 from keras.preprocessing import sequence
  11 from keras.optimizers import Adam
  12 from keras import backend as K
  13 from keras import regularizers
  14 from keras.layers.core import Dense, Dropout, Activation
  15 from keras.layers.embeddings import Embedding
  16 from keras.layers.recurrent import LSTM, GRU
  17 from keras.layers.convolutional import Conv1D
  18 from keras.layers.pooling import MaxPooling1D
  19 from google.cloud import translate_v2
  20 from keras.utils import np_utils
  21 from googletrans import Translator
  22 import matplotlib.pyplot as plt
  23 import pandas as pd
  24 import re
```

```
[ ] 1 ##### GLOBAL VARIABLES #####
    2 #Filenames
    3 Masterdir = ''
    4 translated_data = 'translated_data.txt'
    5 embeddings_dict = {}
    6 check_dict = {}
    7 max_length = 50
```

```
[ ] 1 #LSTM Model Hyperparameters
    2 lstm_parameters_size = 128
    3 # Training Hyperparameters
    4 batch_size = 64
    5 num_epochs = 50
    6 num_classes = 3
    7 fraction_test_set = 0.2
```

```
[ ] 1 def token(sentence, remove_vowels=False, remove_repeat=False, minchars=2):
    2     tokens = []
    3     #for t in re.findall("[A-Z]{2,}(![a-z])|[A-Z][a-z]+(=[A-Z])|[\w]+",sentence.lower()):
    4     for t in re.findall("[a-zA-Z]+",sentence.lower()):
    5
    6         if len(t)>=minchars:
    7             if remove_vowels:
    8                 t=removeVowels(t)
    9             if remove_repeat:
    10                 t=removeRepeat(t)
    11             tokens.append(t)
    12     return tokens
```

```
[ ] 1 def load_embeddings():
    2     """
    3     Purpose -> It Loads the pretrained GloVe Word Embeddings
    4     Input    -> No Arguments
    5     Output   -> Loads the Embeddings into Embeddings_dict
    6     """
    7     f = open("glove.6B.100d.txt", 'r', encoding="utf-8")
    8
    9     for line in f:
    10         values = line.split()
    11         word = values[0]
    12         vector = np.asarray(values[1:], "float32")
    13         embeddings_dict[word] = vector
    14         check_dict[word] = 1
    15
```

```
[ ] 1 def save_data(Masterdir,data_name, data, file_name):
    2     """
    3     Purpose -> Saves the data along with labels as data_name in the folder Masterdir.
    4     Input    -> Data, Folder location where and Data is stored in file File_name
    5     Output   -> Nil
    6     """
    7     h5f = h5py.File(Masterdir+Modeldir+file_name+'.h5', 'a')
    8     h5f.create_dataset(data_name, data=data)
    9     h5f.close()
```

```
[ ] 1 def save_model(Masterdir,filename,model):
2     """
3     Purpose -> Saves the model along with labels as data_name in the folder Masterdir.
4     Input    -> Keras model along with the location of file where it is to be stored.
5     Output   -> Nil
6     """
7     #Referred from:- http://keras.io/getting-started/faq/#how-can-i-save-a-keras-model
8     model.save(Masterdir+'LSTM_'+filename+'_model.h5')
9     model.save_weights(Masterdir+'LSTM_'+filename+'_weights.h5')
```

```
[ ] 1 def get_data(Masterdir, filename, data_name):
2     """
3     Purpose -> It is used to get data stored in an h5py file.
4     Input    -> Filename, folder location and the key name in h5py file.
5     Output   -> Extracted Data
6     """
7     h5f = h5py.File(Masterdir+filename+'.h5', 'r')
8     return h5f[data_name]
```

```
[ ] 1 def get_f1(y_true, y_pred): #taken from old keras source code
2
3     true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
4     possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
5     predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
6     precision = true_positives / (predicted_positives + K.epsilon())
7     recall = true_positives / (possible_positives + K.epsilon())
8     f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
9     return f1_val
```

```
[ ] 1 def evaluate_model(X_test,y_test,model,batch_size,num_classes):
2     """
3     Purpose -> It is used to evaluate the model on the testing data.
4     Input    -> Testing data and its corresponding label, trained model.
5     Output   -> Nil
6     """
7     #Evaluate the accuracies
8     score, acc = model.evaluate(X_test, y_test, batch_size=batch_size)
9     print('Test score:', score)
10    print('Test accuracy:', acc)
```

```
[ ] 1 def print_training_losses(history):
2     """
3     Purpose -> To Plot a graph of Loss versus Epochs during Training
4     Input    -> Keras history variable that contain information related to losses.
5     Output   -> Nil
6     """
7     loss_train = history.history['loss']
8     loss_val = history.history['val_loss']
9     epochs = range(1,num_epochs+1)
10    plt.plot(epochs, loss_train, 'g', label='Training loss')
11    plt.plot(epochs, loss_val, 'b', label='validation loss')
12    plt.title('Training and Validation loss')
13    plt.xlabel('Epochs')
14    plt.ylabel('Loss')
15    plt.legend()
16    plt.show()
```



```

1  def create_embedding_vectors(Masterdir,filename):
2      """
3      Purpose -> Get Word Embeddings and pad the sequence
4      Input   -> Data file containing the list of words
5      Output  -> A feature matrix representing embedding vectors of words.
6      """
7      f=open(Masterdir+Datadir+filename,'r', encoding='utf-8')
8      lines = f.read().split('\n')
9
10     temp = np.random.randn(100)
11     embeddings = []
12     for line in lines:
13         token_lines = token(line)
14         embedding_vector = []
15         for words in token_lines:
16             value = check_dict.get(words)
17             if(value != None):
18                 vector = embeddings_dict.get(words)
19                 #print("vector shape: ",vector.shape)
20                 embedding_vector.append(vector)
21             else:
22                 embedding_vector.append(temp)
23
24         size = len(embedding_vector)
25         for i in range(max_length-size):
26             embedding_vector.append(temp)
27         embedding_vector = np.asarray(embedding_vector)
28         embeddings.append(embedding_vector[0:max_length])
29
30     embeddings = np.asarray(embeddings)
31     return embeddings

```

```

[ ] 1  def print_training_accuracy(history):
2      """
3      Purpose -> To Plot a graph of Accuracy versus Epochs during Training
4      Input   -> Keras history variable that contain information related to Model's Accuracy.
5      Output  -> Nil
6      """
7      acc_train = history.history['accuracy']
8      acc_val = history.history['val_accuracy']
9      epochs = range(1, num_epochs+1)
10     plt.plot(epochs, acc_train, 'g', label='Training accuracy')
11     plt.plot(epochs, acc_val, 'b', label='validation accuracy')
12     plt.title('Training and Validation accuracy')
13     plt.xlabel('Epochs')
14     plt.ylabel('Accuracy')
15     plt.legend()
16     plt.show()

```

```

1 def RNN_Model_Helper(X_train,y_train,hyperparameters):
2
3     lstm_parameters_size = hyperparameters[6]
4     # Training hyperparameters
5     batch_size = hyperparameters[7]
6     nb_epoch = hyperparameters[8]
7     num_classes = hyperparameters[9]
8     fraction_test_set = hyperparameters[10]
9
10    #Train & Validation data splitting
11    X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, train_size=1-fraction_test_set,
12
13    #Build the sequential model
14    print('Build model...')
15    model = Sequential()
16
17    model.add(LSTM(lstm_parameters_size, return_sequences=True))
18    #model.add(LSTM(lstm_parameters_size, dropout=0.3, recurrent_dropout=0.3, return_sequences=True))
19    model.add(LSTM(lstm_parameters_size, return_sequences=False))
20    model.add(Dense(num_classes))
21    model.add(Activation('softmax'))
22
23    #Adamax Optimizer is used and loss model is categorical crossentropy loss
24    model.compile(loss='categorical_crossentropy',
25                  optimizer='adamax',
26                  metrics=['accuracy'])
27
28    print('Train...')
29    #Model is trained for 50 epochs and shuffled after every epoch for training data, model is validated on
30    history = model.fit(X_train, y_train, batch_size=batch_size, shuffle=True, epochs=nb_epoch, validation_data=(X_valid, y_valid))
31    return model,history

```

```

[ ] 1 if __name__ == '__main__':
2
3     #Master function
4     print('Loading GloVe Embeddings...')
5     load_embeddings()
6     print('GloVe Embeddings Loaded Successfully...')
7     print('Creating Embedding Vectors...')
8     embeddings = create_embedding_vectors(Masterdir,translated_data)
9     print(embeddings[0][max_length-1])
10    print('Embedding Vectors Created...')
11    save_data(Masterdir,"Padded_Embeddings", embeddings, "Padded_Embeddings")
12
13    X_train = get_data(Masterdir, "Padded_Embeddings", "Padded_Embeddings")
14    X_train = np.array(X_train)
15    y_train = get_data(Masterdir, "y_train", "y_train")
16    y_train = np.asarray(np_utils.to_categorical(y_train, num_classes))[0:X_train.shape[0],:]
17    #y_train = np.asarray(y_train).flatten()[0:X_train.shape[0]]
18    print(y_train.shape)
19    X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, train_size=1-fraction_test_set, fraction_test_set=fraction_test_set)
20
21    save_data(Masterdir,"X_train", X_train, "Train_Data")
22    save_data(Masterdir,"X_test", X_test, "Test_Data")
23    save_data(Masterdir,"y_train", y_train, "Train_Data")
24    save_data(Masterdir,"y_test", y_test, "Test_Data")
25
26    print('Creating LSTM Network...')
27    model,history = RNN_Model_Helper(deepcopy(X_train),deepcopy(y_train),[MAX_FEATURES, max_length, embedding_size, batch_size, nb_epoch, num_classes, fraction_test_set])

```

```
save_data(Masterdir,"X_train", X_train, "Train_Data")
save_data(Masterdir,"X_test", X_test, "Test_Data")
save_data(Masterdir,"y_train", y_train, "Train_Data")
save_data(Masterdir,"y_test", y_test, "Test_Data")

print('Creating LSTM Network...')
model,history = RNN_Model_Helper(deepcopy(X_train),deepcopy(y_train),[MAX_FEATURES, max_length,

print('Plot Training/Validation Curves...')
print_training_losses(history)
print_training_accuracy(history)

save_model(Masterdir, "Keras", model)
print('Evaluating model...')
evaluate_model(X_test,deepcopy(y_test),model,batch_size,num_classes)
model.summary()
```