Emotion Detection and Recognition via Text

PROJECT REPORT
submitted towards the partial fulfillment of the
requirements for the award of the degree
of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

SANYAM RAJPAL
15116051
AMAN KUMAR SINGLA
15116006

Under the guidance of:

Dr. DEBASHIS GHOSH

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE – 247667 (INDIA)
May, 2019

STUDENT'S DECLARATION

We declare that the work presented in this report with title "**Emotion Detection and Recognition via Text**" towards the fulfillment of the requirement for the award of the degree of **Bachelor of Technology** in **Electronics & Communication Engineering** submitted in the **Department of Electronics & Communication Engineering**, **Indian Institute of Technology, Roorkee**, India is an authentic record of our own work carried out during the period **from August 2018 to May 2019** under the supervision of **Dr. Debashis Ghosh**, Professor, Dept. of ECE, IIT Roorkee. The content of this report has not been submitted by me for the award of any other degree of this or any other institute.

DATE : ..... SIGNED: .....

PLACE: ..... (AMAN KUMAR SINGLA)
ENROL. NO.: 151160007

DATE : ..... SIGNED: .....

PLACE: ..... (SANYAM RAJPAL)
ENROL. NO.: 15116051

SUPERVISOR'S CERTIFICATE

This is to certify that the statement made by the candidates is correct to the best of my knowledge and belief.

DATE : .....

SIGNED: .....

(DR. DEBASHIS GHOSH)
PROFESSOR
DEPT. OF ECE, IIT ROORKEE

ACKNOWLEDGEMENTS

I sincerely thank my B.Tech. Project advisor **Dr. Debhashis Ghosh** for his constant guidance and support throughout the project.

I am thankful to all the research scholars of Sponsored Lab, Department of ECE for helpful discussions at every stage of the project. I thank **Department of Electronics and Communication Engineering, IIT Roorkee** for providing the lab and other resources required for carrying out my project work.

Overall, working on this project has been a knowledgeable experience and provided me with a great experience in the area of Communication & Signal Processing.

ABSTRACT

This project addresses the problem of emotion detection and recognition via text. The goal of this project was to recognise multiple emotions from sarcastic comments. Sentiment analysis can predict various emotions attached to the text.

We first use deploy our models on binary emotions(good or bad). We use the Recurrent Neural Network to predict binary emotions of the text, that is. good or bad. After tuning the model hyperparameters and adding the layers in an orderly fashion to the RNN, we get an accurate model. Then, we use FastText as an alternative to RNN which gives similar accuracy with almost half the training time.

Then, we move on to Convolutional Neural Network, for which expand the emotions to a set of 5: neutral, happy, sad, anger and hate. After attaining high accuracy for the same, we decide to move on from simple sentences to tweets which have sarcastic comments, Singlish slangs and multiple emotions. After, adjusting the layers and tuning the hyperparameters, we manage to improve accuracy.

Sarcastic comments and incomplete sentences as a dataset is a little less explored part of emotion recognition via text and handling such datasets is still a great challenge in itself.

TABLE OF CONTENTS

LIST OF TABLES

TABLE **Page**

List of Figures

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Motivation

Emotion detection and recognition have been worked upon but a significant amount of the work focused on using human speech and facial expressions as the datasets. We decided to use text data and achieving high accuracy with a significant drop in training time. Besides, the model was trained on sarcastic comments and incomplete sentences.

From the experience we gained while building and training the previous model on a simpler dataset, we managed to train a RNN on Twitter datasets which comprised of sarcastic comments, multiple emotions and some Singlish language slang words. It can be used for preventing people from falling victims to fake news, hoaxes and disinformation.

Besides, a system can be designed to indicate to readers that a particular article cannot be relied upon, that is the model is illegitimate. Twitter is a popular social media website for sharing important information, its legitimacy is important to all users. The warning signs are an indication of people reacting to the news with a negative spectrum of offensive comments, confusion and a wide spectrum of emotions.

1.2 Problem Definition

The problem statement is to develop a deep learning model which can detect and recognise **multiple emotions** from the text. The model must be reliable with adequate accuracy and should be efficient enough so that a real-time implementation can be obtained. The model should be able to accurately recognise the given words for which it is trained on any individual. Besides, the model should be efficient in terms of training time.

After achieving the above on a dataset of simple sentences, the same should be achieved on a dataset having **sarcastic comments**.

1.3 Thesis Organization

Chapter 1 gives an introduction to the problem and the need for an accurate model for handling datasets having sarcasm using deep learning.

Chapter 2 discusses the literature review done to get knowledge about the state of the art work done in the field.

Chapter 3 discusses the types of datasets used and its features. It discusses the hardships faced while extracting features from the data. Besides, it tells about the preprocessing involved in extracting features from the datasets.

Chapter 4 discusses the functions and techniques used while making and training the model. Besides, it lays importance on the architecture of the models built and trained.

Chapter 5 discusses the results accomplished using descriptive diagrams helping in understanding the training procedure and the testing results.

Chapter 6 concludes this thesis and provides the inferences drawn from this project. Besides, it gives the scope of future research in this area.

CHAPTER 2

LITERATURE REVIEW

2.1 Overview

This chapter shows the concepts learned before we laid hands on real-time dataset and the techniques we learned to implement deep neural networks for the same. The learning we did while working on this helped us in filtering out the details.

2.2 Literature

There are many papers written on sentiment analysis for the domain.

(Ache and Lee 2008) gives an overview of supposition investigation [5] In a paper, Jansen has broken down the business effect of social interceding innovation, microblogging . Generally speaking, content grouping utilizing AI is a well-contemplated field (Manning and Schuetze 1999).[3]

There is an astounding work on the impacts of different AI methods, for example, Naive Bayes, Maximum Entropy, SVM in the film surveys domain. (Pang and Lee 2002).[4] They had the capacity to accomplish an exactness of 82.9% utilizing SVM and a unigram model. Work (Read, 2005) has been done in utilizing emojis as names for positive and slant. This is extremely associated with Twitter in light of the fact that numerous clients have emojis in their tweets.

In the late year, Recurrent Neural Networks and Long Short term Memory systems have had the capacity to encode the content successions in all respects effectively by holding the long haul transient conditions. Likewise, headways in generative demonstrating by utilizing Variational Autoencoders and Generative Adversarial Networks have started to produce profoundly practical pictures. [1]

Analysts have additionally dealt with recognizing supposition in content. (Turney 2002) presents a straightforward calculation, called semantic introduction, for distinguishing notion.[8]

Instead of legitimately joining the microblogging highlights into assessment classifier preparing, Speriosu et al. [2] built a chart that has a portion of the microblogging highlights, for example, hashtags and emojis together with clients, tweets, word unigrams and bigrams as its hubs which are associated dependent on the connection presence among them (e.g., users are associated with tweets they made; tweets are associated with word unigrams that they contain and so forth.).

They at that point connected a mark engendering strategy where assumption names were spread from a little arrangement of hubs seeded with some underlying name data all through the diagram. They guaranteed that their mark spread strategy outflanks MaxEnt prepared from uproarious names and got an exactness of 84.7% on the subset of the Twitter assessment test set from[6].

The gave tweets were a blend of words, emojis, URLs, hashtags, client notices, and images. Before preparing they pre-process the tweets to make it reasonable for sustaining into models. The creators executed a few AI calculations like Naive Bayes, Maximum Entropy, Decision Tree, Random Forest, XGBoost, SVM, Multi-Layer Perceptron, Recurrent Neural systems and Convolutional Neural Networks to characterize the extremity of the tweet.[9]They utilized two sorts of highlights specifically, unigrams and bigrams for order and sees that increasing the component vector with bigrams improved the exactness. When the element has been extricated it was spoken to as either a scanty vector or a tick vector. It has been seen that nearness in the meagre vector portrayal recorded a superior act than recurrence.

Neural strategies performed superior to different classifiers as a rule. Our best LSTM model accomplished an exactness of 83.0% on Kaggle while the best CNN model accomplished 83.34%. The model which utilized highlights from our best CNN model

and arranges utilizing SVM performed somewhat superior to just CNN. They utilized a group strategy taking a lion's share vote over the expectations of 5 of their best models accomplishing a precision of 83.58%.[10]

The expansion of microblogging locales like Twitter offers an unparalleled opening to frame and utilize approaches and innovations that look and dig for feelings. The work introduced in this paper determines a methodology for supposition examination on Twitter information. To unlock the assumption, we extricated the important information from the tweets included the highlights. The general tweet estimation was then determined to utilize a model that displayed in this report. This work is exploratory in nature and the model assessed is a primer model. The models demonstrated that forecast of content notion is a non-paltry undertaking for AI. A ton of preprocessing is required just to have the capacity to run a calculation. The fundamental issue for the assumption the investigation is to make the machine portrayal of the content.[11]

Straightforward pack of-words was certainly not enough to acquire fulfilling results, hence a ton of extra highlights was made based on the normal sense (number of emojis, outcry marks, number of the question mark and so forth).[12] The authors believed that a slight improvement in grouping precision for the given preparing dataset could be grown, yet since it included exceptionally skewed information (modest number of negative cases), the distinction will be likely in the request of a couple of percents. What could improve characterization results will be to include a lot of extra models (increment preparing dataset), on the grounds that given 5971 precedents obviously do exclude all arrangement of words utilized, further - a great deal of feeling communicating data surely is missing.

2.3 Summary

This chapter presents the things we learned while before doing the project and while working on the project when necessary. It helped us in understanding the concepts of deep learning thoroughly.

CHAPTER 3

DATASET PROCESSING

3.1 Overview

This chapter deals with the datasets we worked upon during the tenure of the project and the procedure done for generating those datasets.

3.2 IMDB Dataset

This is a dataset for double conclusion arrangement containing generously a greater number of information than past benchmark datasets. We give a lot of 25,000 profoundly polar motion picture surveys for preparing, and 25,000 for testing. There is extra unlabeled information for use also. Crude content and as of now handled pack of words designs are given.

3.3 Glove Vector Dataset

The GloVe is an unsupervised learning calculation for getting vector portrayals for words. Preparing is performed on accumulated worldwide word-word co-event measurements from a corpus, and the subsequent portrayals exhibit fascinating straight substructures of the word vector space.

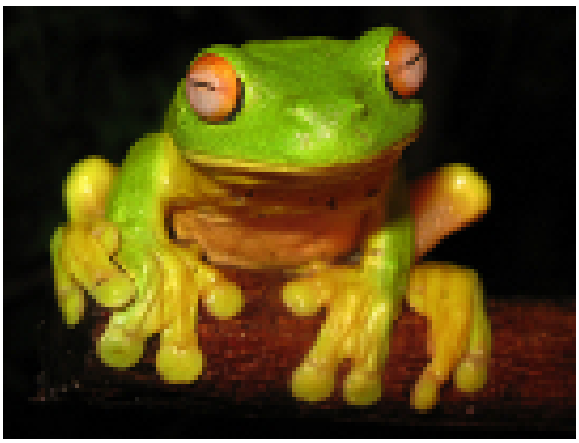


Figure 3.1: Litoria

Figure 3.2: Leptodactylidae

Figure 3.3: Rana

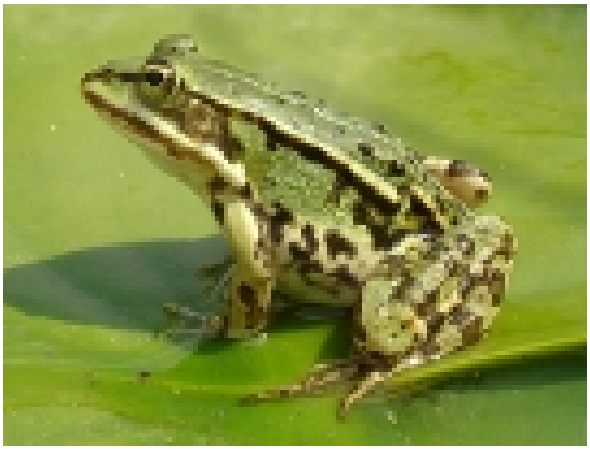


Figure 3.4: Eleutherodactylus

3.3.1 Techniques Used

3.3.1.1 Nearest Neighbor

The Euclidean separation (or cosine likeness) between two-word vectors gives a viable strategy for estimating the phonetic or semantic closeness of the comparing words. Now and again, the closest neighbours as indicated by this measurement uncover uncommon however important words that lie outside a normal human's vocabulary.

3.3.2 Linear Substructures

The closeness estimations used for nearest neighbour appraisals produce alone scalar that assesses the relatedness of two words. This ease can be dangerous since two given

words frequently show progressively multifaceted associations that can be gotten by a single number. For example, a man may be seen as the like woman in that the two words depict individuals; of course, the two words are normally seen as opposite energies since they highlight a fundamental turn along which individuals differentiate from one another. So as to get in a quantitative manner the subtlety basic to see the man from lady, it is basic for a model to relate in excess of a solitary number to the word pair. A trademark and direct contender for an upgraded game plan of discriminative numbers is the vector distinction between the two-word vectors. The GloVe is composed all together that such vector contrasts get in any case much as could be typical the hugeness shown by the juxtaposition of two words.

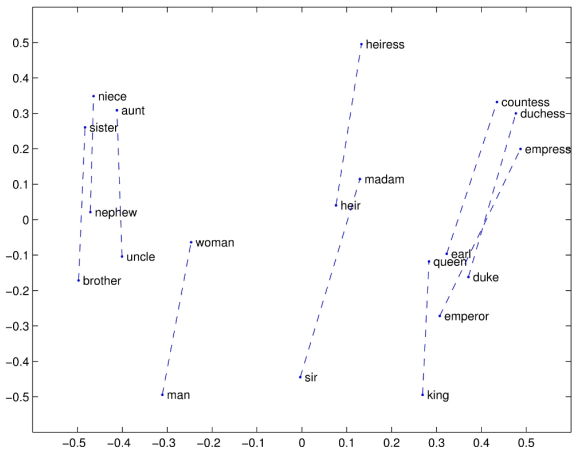


Figure 3.5: man - woman

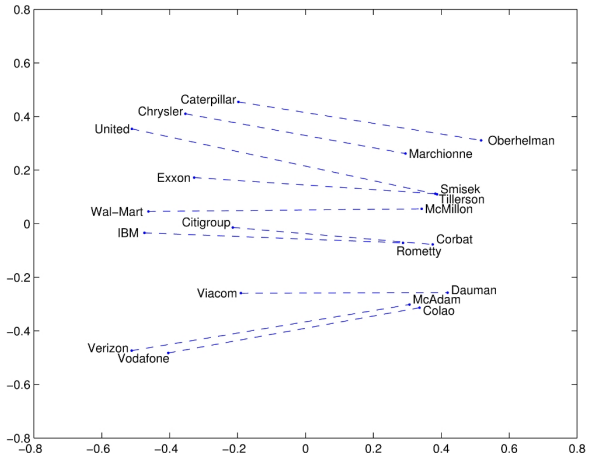


Figure 3.6: company - ceo

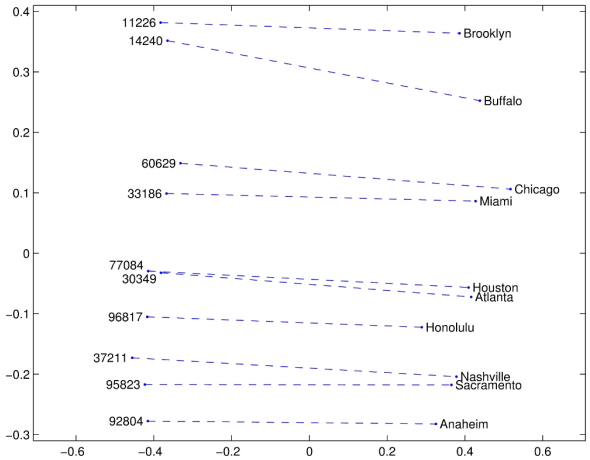


Figure 3.7: city - zip code

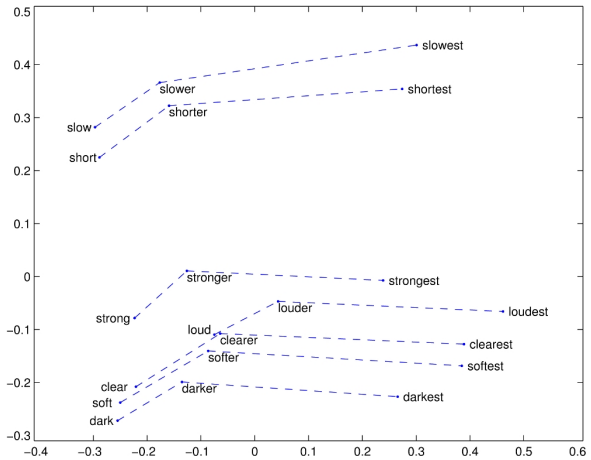


Figure 3.8: comparative - superlative

The concealed thought that perceives man from woman, for instance, sex or sexual introduction may be indistinguishably demonstrated by various other word sets, for instance, ruler and ruler or kin and sister. To express this discernment numerically, we may expect that the vector contrasts man - woman, ruler, and kin - sister may all be commonly proportional. This property and other intriguing precedents can be found in the above game plan of portrayals.

3.4 Twitter Dataset

The first dataset is contained 40,000 tweets grouped into 13 feeling classes. Nonetheless, past creators have portrayed that few of those classes were in certainty incredibly comparative, and rehashed endeavours to re-mark the information just outcome in 72% understanding. Henceforth, we settle on the choice to join a few of those classes into five last classes. These five classes are likewise equivalent to in Bouazizi and Ohtsuki (2017), with the nonattendance of the "mockery" and "love" classes (Binary to Multi-Class Classification) or "fun" and "love" classes (Pattern-Based Approach for Multi-Class Sentiment Analysis).

We likewise pulled information from the Twitter utilizing Twitter API as extra preparing information. The tweets are characterized by their very own hashtags - for instance, "happy". We feel that hashtags ought to be an obviously decent (however a long way from impeccable) portrayal of the slant of the tweet. While it is conceivable for somebody to tweet something like "Uh, I got 90 for A dimensions sad", this is a little minority and can be taken to be a factual commotion, which may have the additional advantage of lessening over-fitting of preparing information. Being tweets, the content is short, casual, and ranges a wide scope of subjects. We tried our model on an assortment of real remarks made on Reddit and Facebook by Singaporeans in "Singlish" and in our extraordinary, nearby mix of online slang.Our outcomes drove us to two ends:

- Singlish was a little test than recently anticipated. It is conceivable that some Singlish is caught in the GloVe vectors amid preparing. Also, it may be the case that the remainder of the remark as of now gives the model adequate insight to the feeling with the end goal that the concealed part (Singlish) isn't pivotal in characterization.
- Sacarsm was a serious issue. Our model did not learn mockery, and along these

lines misclassifies remarks which are wry. In a creation setting, we propose utilizing another separate model to distinguish mockery and to dispose of the outcomes from this model should the other model identify an abnormal state of mockery.

- Numerous remarks express various feelings. Henceforth, it may be advantageous to consider the multi-mark arrangement to, for instance, name a remark as both irate and miserable.

3.5 Binary Emotion Dataset Handling

One of the principle ideas of TorchText is the Field. These characterize how your information ought to be prepared. In our feeling order task, the information comprises of both the crude string of the survey and the supposition, either "pos" or "neg". The parameters of a Field indicate how the information ought to be handled. We utilize the TEXT field to characterize how the survey ought to be prepared, and the LABEL field to process the conclusion.

Our TEXT field has tokenized =’spacy’ as a contention. This characterizes the "tokenization" (the demonstration of part the string into discrete "tokens") ought to be finished utilizing the spacy tokenizer. On the off chance that no tokenize contention is passed, the default is just part the string into spaces. Mark is characterized by a LabelField, an exceptional subset of the Field class explicitly utilized for taking care of names. We will clarify the datatype contention later. We likewise set the irregular seeds for reproducibility.

Another convenient element of TorchText is that it has support for regular datasets utilized in characteristic language preparing (NLP). The code naturally downloads the IMDb dataset and parts it into the authoritative train/test parts as torchtext.datasets objects. It forms the information utilizing the Fields we have recently characterized. The IMDb dataset comprises of 50,000 film audits, each set apart just like a positive or negative survey.

The IMDb dataset just has train/test parts, so we have to make an approval set. We can do this with the .split() technique. Of course, this parts 70/30, anyway by passing a split-ratio contention, we can change the proportion of the split, for example, a split-ratio of 0.8 would mean 80% of the precedents make up the preparation set and 20% make up

the approval set. We likewise pass our arbitrary seed to the random-state contention, guaranteeing that we get a similar train/approval split each time.

Next, we need to construct a vocabulary. This is a viable a look-into a table where each exceptional word in your informational collection has a comparing list (a whole number). We do this as our AI model can't work on strings, just numbers. Each record is utilized to develop a one-hot vector for each word. A one-hot vector is where the majority of the components are 0, aside from one, which is 1, and dimensionality is the all-out number of remarkable words in your vocabulary, ordinarily signified by V.

word	index	one-hot vector
I	0	[1, 0, 0, 0]
hate	1	[0, 1, 0, 0]
this	2	[0, 0, 1, 0]
film	3	[0, 0, 0, 1]

The quantity of one of a kind words in our preparation set is more than 100,000, which implies that our one-hot vectors will have more than 100,000 measurements! This will make preparing moderate and perhaps won't fit onto your GPU (in case you're utilizing one).

There are two different ways to successfully chop down our vocabulary, we can either just take the top n most regular words or disregard words that seem not as much as m times. We'll do the previous, just keeping the best 25,000 words.

What do we do with words that show up in precedents yet we have cut from the vocabulary? We supplant them with a unique obscure or <unk> token. For instance, if the sentence was "This film is incredible and I adore it" yet "love" was not in the vocabulary, it would turn into "This film is extraordinary and I <unk> it". The code fabricates the vocabulary, just keeping the most widely recognized maxsize tokens.

When testing any AI framework you would prefer not to take a gander at the test set in any capacity. We do exclude the approval set as we need it to mirror the test set however much as could reasonably be expected. The vocab estimate 25002 and not 25000. One of the extra tokens is the <unk> token and the other is a <pad> token.

When we feed sentences into our model, we feed a group of them at any given moment, for example, more than each one in turn, and all sentences in the cluster should be a similar size. Subsequently, to guarantee each sentence in the group is a similar size, any shorter than the longest inside the bunch are cushioned.

sent1	sent2
I	This
hate	film
this	sucks
film	<pad>

We can likewise see the most widely recognized words in the vocabulary and their frequencies. The last advance of setting up the information is making the iterators. We repeat over these in the preparation/assessment circle, and they return a clump of models (ordered and changed over into tensors) at every cycle.

We'll utilize a BucketIterator which is an uncommon kind of iterator that will restore a clump of models where every precedent is of a comparative length, limiting the measure of cushioning per precedent.

We likewise need to put the tensors returned by the iterator on the GPU (in case you're utilizing one). PyTorch handles this utilizing light. a gadget, we at that point pass this gadget to the iterator. For multiple emotions, we never again expressly need to make the bi-grams and add them as far as possible of the sentence.

3.6 Word Embedings

We have in all respects quickly secured how word embeddings (otherwise called word vectors) are utilized in the instructional exercises. In this index, we'll have a more intensive take a gander at these embeddings and discover a few (ideally) intriguing outcomes. Embeddings change a one-hot encoded vector (a vector that is 0 in components aside from one, which is 1) into a lot littler measurement vector of genuine numbers. The one-hot encoded vector is otherwise called a meagre vector, while the genuine esteemed

vector is known as a tick vector.

The key idea in these word embeddings is that words that show up in comparative settings seem adjacent in the vector space, for example, the Euclidean separation between these two-word vectors is little. By setting here, we mean the encompassing words. For instance, in the sentences "I bought a few things at the shop" and "I acquired a few things at the store" the words 'shop' and 'store' show up in a similar setting and in this manner ought to be near one another in vector space.

You may have additionally found out about word2vec. word2vec is a calculation (really a cluster of calculations) that computes word vectors from a corpus. In this addendum we use GloVe vectors, GloVe being another calculation to ascertain word vectors. In the event that you need to know how Glove attempts to check the site here[18]. In PyTorch, we use word vectors with the nn. Embedding layer, which takes a [sentence length, cluster size] tensor and changes it into a [sentence length, clump measure, inserting dimensions] tensor.

In instructional exercise 2 onwards, we additionally utilized pre-prepared word embeddings (explicitly the GloVe vectors) given by TorchText. These embeddings have been prepared on a colossal corpus. We can utilize these pre-prepared vectors inside any of our models, with the possibility that as they have effectively taken with regards to each word they will give us a superior beginning stage for our oath vectors. This normally prompts quicker preparing time or potentially improved precision. In this addendum we won't prepare any models, rather, we'll be taking a gander at the word embeddings and finding a couple of fascinating things about them.

3.6.1 Vector Handling

We may need to re-utilize the embeddings we have prepared here with another model. To do this begin with, we'll load the GloVe vectors. The name field indicates what the vectors have been prepared on, here the 6B implies a corpus of 6 billion words. The diminish contention determines the dimensionality of the word vectors. GloVe vectors are accessible in 50, 100, 200 and 300 measurements. There is additionally a 42B and 840B glove vectors, be that as it may, they are just accessible at 300 measurements. As appeared, there are 400,000 novel words in the GloVe vocabulary. These are the most widely recognized words found in the corpus the vectors were prepared on. In this arrangement of GloVe vectors, each and every word is lower-case as it were.

This is the authoritative precedent which flaunts this property of word embeddings. Things being what they are, the vector determined from king' minus 'man' gives us a "royalty vector". This is the vector-related with going from a man to his royal counterpart, a king. If we add this "royalty vector" to 'woman', this should make a trip to her illustrious identical, which is a queen! , we'll compose a capacity that will circle through our vocabulary, getting the word and inserting for each word, thinking of them to a content record in a similar organization as our custom embeddings so they can be utilized with TorchText once more.

3.6.2 Freezing and unfreezing embeddings

We're going to prepare our model for 10 ages. Amid the initial 5 ages, we are going to solidify the loads (parameters) of our installing layer. For the last 10 pages, we'll permit our embeddings to be prepared.

Here and there the pre-prepared word embeddings we use will as of now be sufficient and won't should be adjusted with our model. On the off chance that we keep the embeddings solidified, at that point we don't need to ascertain the slopes and update the loads for these parameters, giving us quicker preparing occasions. This doesn't generally apply for the model utilized here, yet we're for the most part covering it to demonstrate how it's finished. Another reason is that if our model has an expansive of parameters it might make preparing troublesome, so by solidifying our pre-prepared embeddings we decrease the number of parameters waiting to be found out.

To solidify the installing loads, we set model.embedding.weight.requiressgrad to False. This will make no slopes be determined for the loads in the installing layer, and along these lines, no parameters will be refreshed when optimizer.step() is called. At that point, amid preparing, we check if FREEZE-FOR (which we set to 5) ages have passed. On the off chance that they have, at that point we set model.embedding.weight.requires-grad to True, revealing to PyTorch that we ought to compute inclinations in the implanting layer and update them with our streamlining agent.

3.6.3 Spelling Mistakes

We'll put their discoveries into code and quickly clarify them, however, to peruse progressively about this, look at the original thread[11] and the related write-up[12]. To begin

with, we have to stack up to a lot bigger vocabulary GloVe vectors, this is because of the spelling botches not showing up in the little vocabulary.

3.6.4 Saving Embeddings

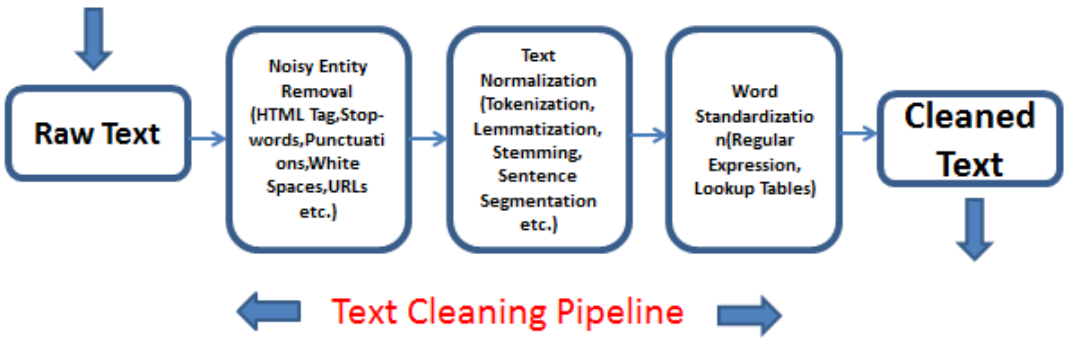
We may need to re-utilize the embeddings we have prepared here with another model. To do this, we'll compose a capacity that will circle through our vocabulary, getting the word and inserting for each word, thinking of them to a content record in a similar organization as our custom embeddings so they can be utilized with TorchText once more.

3.7 Handling Multiple Emotion Tweets

3.7.1 Viterbi Segment

The Viterbi estimation is a dynamic programming computation for discovering them most likely progression of disguised states,Äîconsidered the Viterbi way that results in a game plan of watched events, especially with respect to Markov information sources and covered Markov models.

It is by and by in like manner regularly used in talk affirmation, talk mix, diarization,[6] catchphrase spotting, computational semantics, and bioinformatics. For example, in talk to-content (talk affirmation), the acoustic banner is treated as the watched plan of events, and a string of substance is seen as the "covered reason" of the acoustic banner. The Viterbi count finds the more likely than not string of substance given the acoustic banner.



3.7.2 Hashtag Segment

Crude tweets scratched from twitter, for the most part, result in a boisterous dataset. This is because of the easygoing idea of individuals' use of internet-based life. Tweets have certain uncommon qualities, for example, retweets, emojis, the client makes reference to, and so on which must be reasonably removed. Along these lines, crude twitter information must be standardized to make a dataset which can be effectively learned by different classifiers. We have connected a broad number of pre-preparing ventures to institutionalize the dataset and decrease its size. We initially do some broad pre-handling on tweets which is as per the following.

- URL Users regularly share hyperlinks to different website pages in their tweets. A specific URL isn't significant for content characterization as it would prompt scanty highlights. In this way, we supplant every one of the URLs in tweets with the word URL. The standard articulation used to coordinate URLs.
- Client Mention Every twitter client has a handle related to them. Clients regularly notice different clients in their tweets by @handle. We supplant all client specifies with the word USER-MENTION. The customary articulation used to coordinate client notice.
- Hashtags are unspaced expressions prefixed by the hash image () which is regularly utilized by clients to make reference to a slanting subject on Twitter. We supplant all the hashtags with the words with the hash image. For instance, hello is supplanted by hi. The customary articulation used to coordinate hashtags.
- It is difficult to thoroughly coordinate all the distinctive emojis utilized via web-based networking media as the number is regularly expanding. Be that as it may, we coordinate some basic emojis which are utilized in all respects oftentimes. We supplant the coordinated emojis with either emopos or emoneg relying upon whether it is passing on a positive or a negative feeling.

Retweets are tweets which have just been sent by another person and are shared by different clients. Retweets start with the letters RT. We expel RT from the tweets as it's anything but a significant component for content grouping. The ordinary articulation is used to coordinate retweets. In the wake of applying tweet level pre-handling, we prepared individual expressions of tweets as pursues. A few people send tweets like I am

sooooo happppppy including various characters to accentuate certain words. This is done to deal with such tweets by changing over them so I am so glad. Removing such things is done to deal with words like a shirt and their's by changing over them to the more broad structure shirt and theirs. We characterize a legitimate word as a word which starts with a letter in order with progressive characters being letters in order, numbers or one of speck (.) and underscore.

3.8 Summary

This chapter presented the datasets used and the processes involved in generating them. Besides, various approaches are discussed for handling the datasets to process them and extract features from them.

CHAPTER 4

TRAINING TECHNIQUES AND MODEL ARCHITECTURES

4.1 Overview

This chapter discusses the model used, the steps followed while training the model and the techniques deployed while training the model. The steps are taken in order to optimize the architecture in terms of depth and its ability to extract information.

4.2 RNN for Binary Emotions

The inserting layer is utilized to change our meagre one-hot vector (inadequate as a large portion of the components are 0) into a thick implanting vector (thick as the dimensionality is much littler and every one of the components is genuine numbers). This implanting layer is essentially a solitary completely associated layer. Just as decreasing the dimensionality of the contribution to the RNN, there is the hypothesis that words which have a comparative effect on the assessment of the audit are mapped near one another in this thick vector space. For more data about word embeddings, see here.

The RNN layer is our RNN which takes in our thick vector and the past concealed state h_{t-1}, which it uses to figure the following shrouded state, h_t.

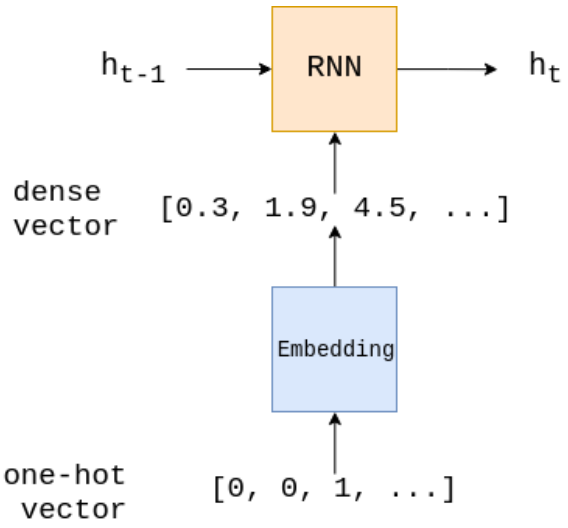


Figure 4.1: RNN for binary motions

At last, the straight layer takes the last concealed state and feeds it through a completely associated layer, f(h_T), changing it to the right yield measurement. The forward strategy is called when we feed precedents into our model.

Each clump, content, is a tensor of size [sentence length, cluster size]. That is a cluster of sentences, each having each word changed over into a one-hot vector. PyTorch helpfully stores a one-hot vector as it's recorded esteem, for example, the tensor speaking to a sentence is only a tensor of the records for every token in that sentence. The demonstration of changing over a rundown of tokens into a rundown of lists is regularly called numericalizing.

The information cluster is then gone through the inserting layer to get installed, which gives us a thick vector portrayal of our sentences. installed is a tensor of size [sentence length, clump estimate, inserting dim] implanted is then bolstered into the RNN. In certain structures you should sustain the underlying concealed state, h_0, into the RNN, anyway in PyTorch, if no underlying shrouded state is passed as a contention it defaults to a tensor of each of the zeros.

The RNN returns 2 tensors, yield of size [sentence length, group estimate, shrouded dim] and covered up of size [1, clump measure, concealed dim]. yield is the connection of the concealed state from each time step while covering up is just the last shrouded state. We check this utilizing the declare articulation. Note the press strategy, which is utilized

to expel an element of size 1.

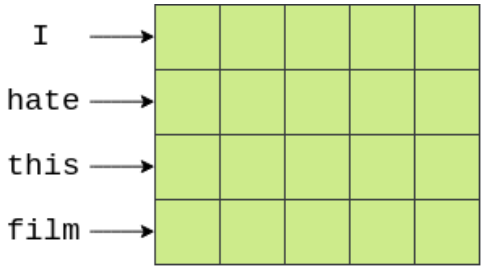
The information measurement is the element of the one-hot vectors, which is equivalent to the vocabulary estimate. The implanting measurement is the span of the thick word vectors. This is for the most part around 50-250 measurements, however, relies upon the extent of the vocabulary.

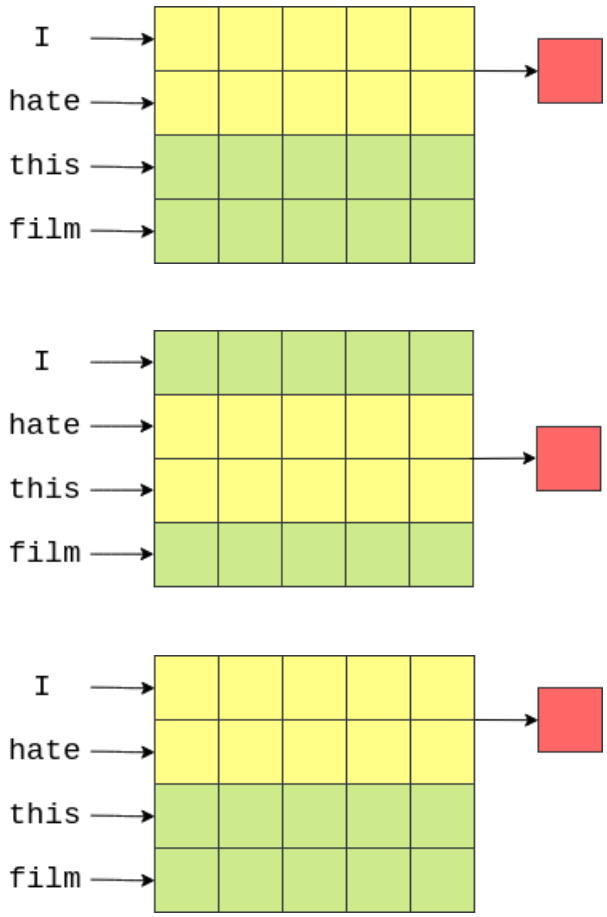
The shrouded measurement is the extent of the concealed states. This is more often than not around 100-500 measurements, yet in addition relies upon components, for example, on the vocabulary estimate, the extent of the thick vectors and the unpredictability of the undertaking.

The yield measurement is typically the number of classes, anyway on account of just 2 classes the yield esteem is somewhere in the range of 0 and 1 and in this manner can be 1-dimensional, for example, a solitary scalar genuine number.

4.3 CNN for Emotions

The first major hurdle is visualizing how CNNs are used for text. Images are typically 2 dimensional whereas text is 1 dimensional. However, we know that the first step in almost all of our previous preprocessing is converting the words into word embeddings. This is how we can visualize our words in 2 dimensions, each word along one axis and the elements of vectors across the other dimension. Consider the 2-dimensional representation of the embedded sentence. We would then be able to utilize a channel that is [n x emb-dim]. This will cover N consecutive words altogether, as their width will be emb-diminish measurements. Consider the picture underneath, with our assertion vectors, are spoken to in green. Here we have 4 words with 5-dimensional embeddings, making a [4x5] "picture" tensor.





A channel that covers two words at any given moment (for example bi-grams) will be [2x5] channel, appeared yellow, and every component of the channel with have a weight-related with it. The yield of this channel (appeared red) will be a solitary genuine number that is the weighted total of all components secured by the channel.

The channel at that point moves "down" the picture (or over the sentence) to cover the following bi-gram and another yield (weighted total) is determined. At long last, the channel moves down again and the last yield for this channel is determined.

For our situation (and in the general situation where the width of the channel approaches the width of the "picture"), our yield will be a vector with number of components equivalent to the stature of the picture (or length of the word) short the tallness of the channel in addition to one, 4-2+1=3 for this situation.

This precedent told the best way to compute the yield of one channel. Our model (and essentially all CNNs) will have loads of these channels. The thought is that each channel

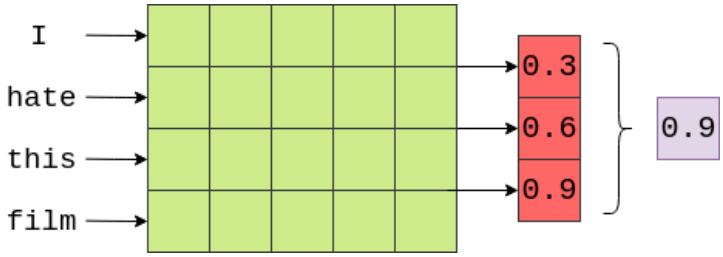
will get familiar with an alternate component to remove. In the above precedent, we are trusting each of the [2 x emb-dim]filters will search for the event of various bi-grams.

In our model, we will likewise have distinctive sizes of channels, statures of 3, 4 and 5, with 100 of every one of them. The instinct is that we will search for the event of various tri-grams, 4-grams and 5-grams that are significant for dissecting the conclusion of film audits.

The following stage in our model is to utilize pooling (explicitly max pooling) on the yield of the convolutional layers. This is like the FastText model where we played out the normal over every one of the word vectors, executed by the F.avg-pool2d work, anyway as opposed to taking the normal over a measurement, we are taking the most extreme incentive over a measurement. Underneath a case of taking the greatest esteem (0.9) from the yield of the convolutional layer on the precedent sentence (not demonstrated is the enactment work connected to the yield of the convolutions).

The thought here is that the greatest esteem is the "most significant" highlight for deciding the slant of the survey, which compares to the "most significant" n-gram inside the audit. How would we know what the "most significant" n-gram is? Fortunately, we don't need to! Through backpropagation, loads of the channels are changed so that at whatever point certain n-grams that are profoundly characteristic of the opinion are seen, the yield of the channel is "high" esteem. This "high" esteem at that point goes through the maximum pooling layer in the event that it is the most extreme incentive in the yield.

As our model has 100 channels of 3 distinct sizes, that implies we have 300 diverse n-grams the model believes are significant. We connect these together into a solitary vector and pass them through a straight layer to foresee the assumption. We can think about loads of this straight layer as "weighing up the proof" from every one of the 300 n-grams and settling on an official choice.



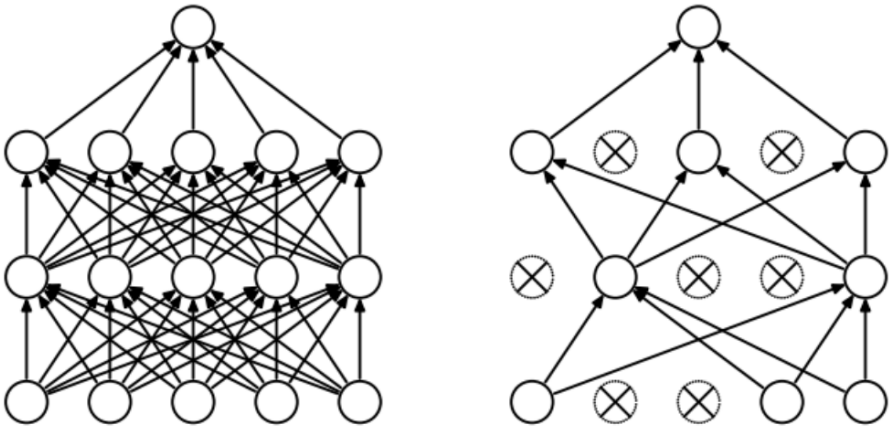
4.4 LSTMs for Twitter Dataset

4.4.1 Dropout

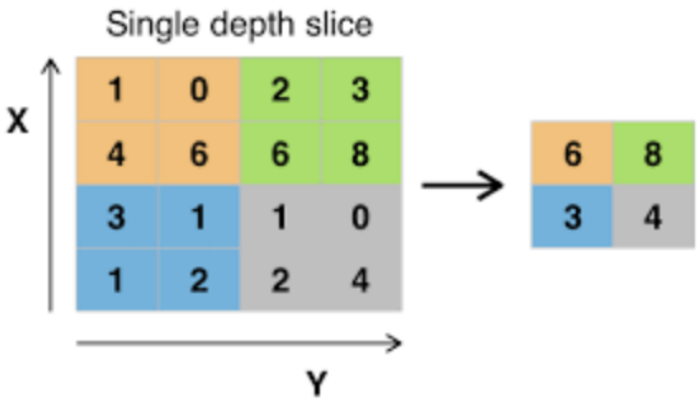
In spite of the fact that we've added enhancements to our model, everyone includes extra parameters. Without going into overfitting into an excessive amount of detail, the more parameters you have in your model, the higher the likelihood that your model will overfit (retain the preparation information, causing a low preparing mistake however high approval/testing blunder, for example, poor speculation to new, concealed precedents).

To battle this, we use regularization. All the more explicitly, we utilize a technique for regularization called dropout. Dropout works by haphazardly dropping out(setting to 0) neurons in a layer amid a forward pass. The likelihood that every neuron is dropped out is set by a hyperparameter and every neuron with dropout connected is considered independently.

One hypothesis concerning why dropout works is that a model with parameters dropped out can be viewed as a "more fragile" (fewer parameters) model. The expectations from all these "flimsier" models (one for each forward pass) get arrived at the midpoint of together within the parameters of the model. Along these lines, your one model can be thought of as an outfit of more fragile models, none of which are over-parameterized and in this way ought not to overfit.



(a) Standard Neural Net (b) After applying dropout.

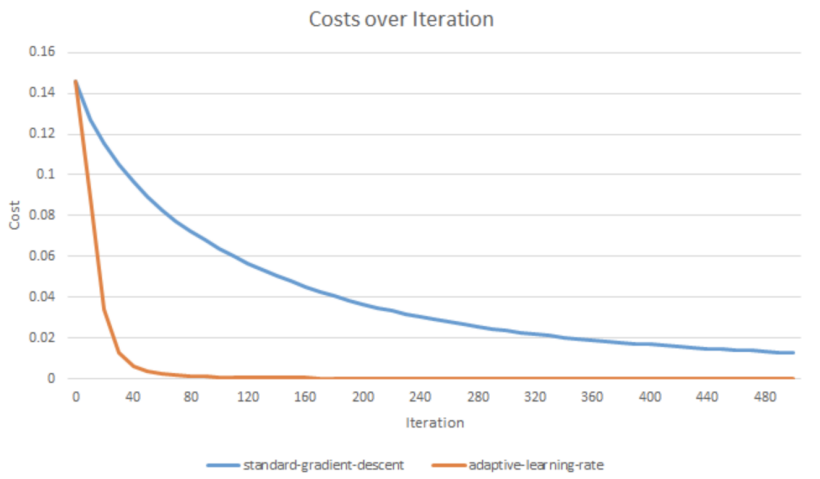


4.4.2 MaxPooling

Max pooling is an example based discretization process. The goal is to down-example an info portrayal (picture, shrouded layer yield network, and so forth.), decreasing its dimensionality and considering suspicions to be made about highlights contained in the sub-areas binned.

4.4.3 Adaptive Learning Rate

Adaptive learning rate technique keeps the learning rate the same as the previous value as long as the training loss keeps on decreasing. As soon as the training loss stops decreasing, the method downscales the learning rate by a constant and repeats the similar procedure until the decrease of training loss completely stops. Adaptive learning rate helps the model learn faster.



4.4.4 Loss Function

Softmax is a capacity which squashes a vector in the range (0, 1) and all the subsequent components mean 1. It is connected to the yield scores **s**. As components speak to a class, they can be deciphered as class probabilities. The cross-entropy Loss is characterized as Where $\mathbf{t_i}$ and $\mathbf{s_i}$ are the ground truth and the Neural Network score for each class $\mathbf{_i}$ in C. As for the most part an enactment work (Softmax) is connected to the scores before the CE Loss calculation, we compose $\mathbf{f(s_i)}$ to allude to the initiations. It is a Softmax enactment in addition to a Cross-Entropy misfortune. On the off chance that we utilize this misfortune, we will prepare a CNN to yield a likelihood over the **C** classes for each picture. It is utilized for multi-class order. In the particular instance of Multi-Class arrangement, the names are one-hot, so just the positive class $\mathbf{C_i}$ keeps its term in the misfortune. There is just a single component of the Target vector t which isn't zero $\mathbf{t_i} = \mathbf{t_p}$. So disposing of the components of the summation which are zero because of target names, we can compose:

$$CE = \frac{1}{M}\sum_{p}^{M} -log\frac{e^{s_{\mathrm{p}}}}{\sum_{j}^{C} e^{s_{\mathrm{p}}}} \tag{4.1}$$

Where each $\mathbf{s_p}$ in M is the CNN score for every positive class. We present a scaling factor **1/M** to make the misfortune invariant to the number of positive classes, which might be the diverse per test.

4.4.5 Optimizer

Adagrad is an enhancer with parameter-explicit learning rates, which are adjusted in respect to how much of the time a parameter gets refreshed amid preparing. The more updates a parameter gets, the littler the learning rate. Adadelta is an increasingly hearty expansion of Adagrad that adjusts learning rates dependent on a moving window of slope refreshes, rather than amassing every single past inclination. Along these lines, Adadelta keeps adapting notwithstanding when numerous updates have been finished. Contrasted with Adagrad, you don't need to set an underlying learning rate for Adadelta.

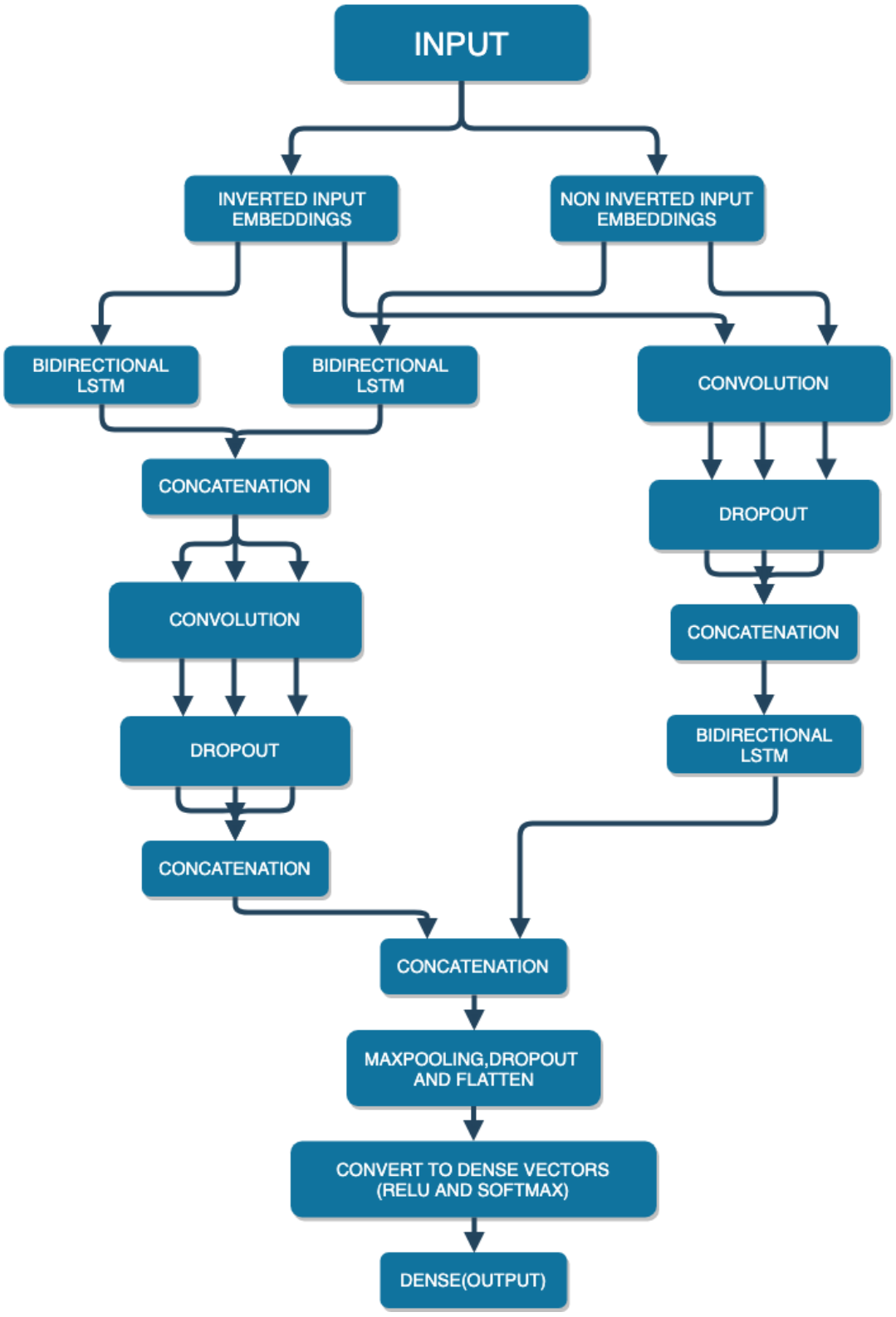


Figure 4.2: Architecture

4.4.6 Architecture

The architecture shown above showcases the model we built for handling the Twitter dataset.

- First, we feed the input embeddings to the BiDirectional LSTM and Convolution layer separately.
- After concatenating the weights from both features extracting models, we apply Maxpooling layers, dropout layers and then flatten the weight matrices.
- Thereafter they are converted to the dense vector using Relu and Softmax and then used as features for training and testing the model.

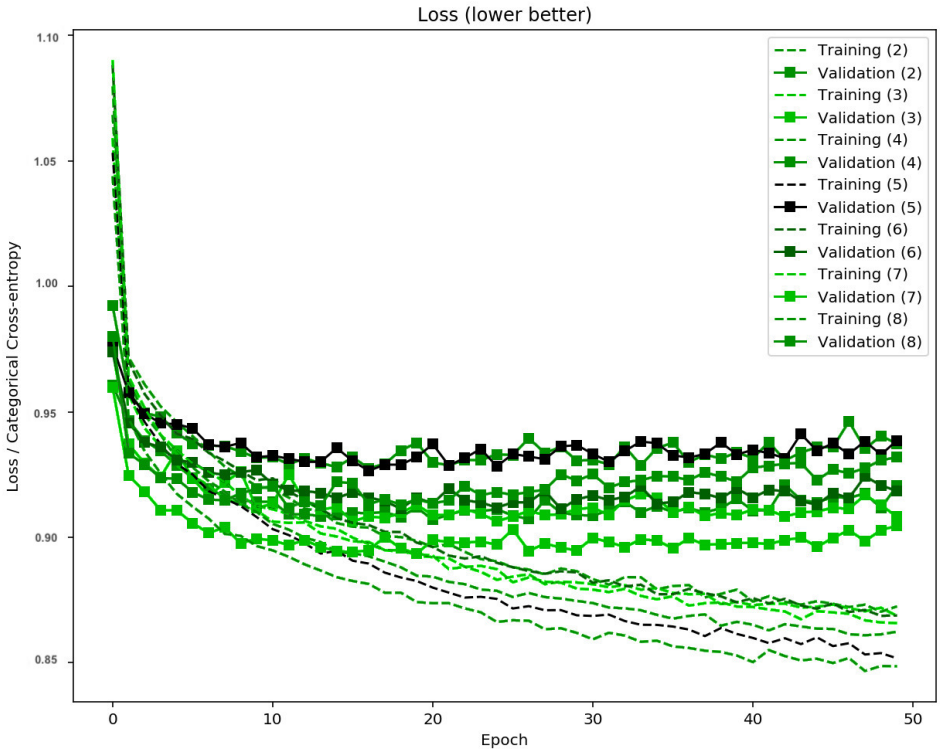
4.5 Summary

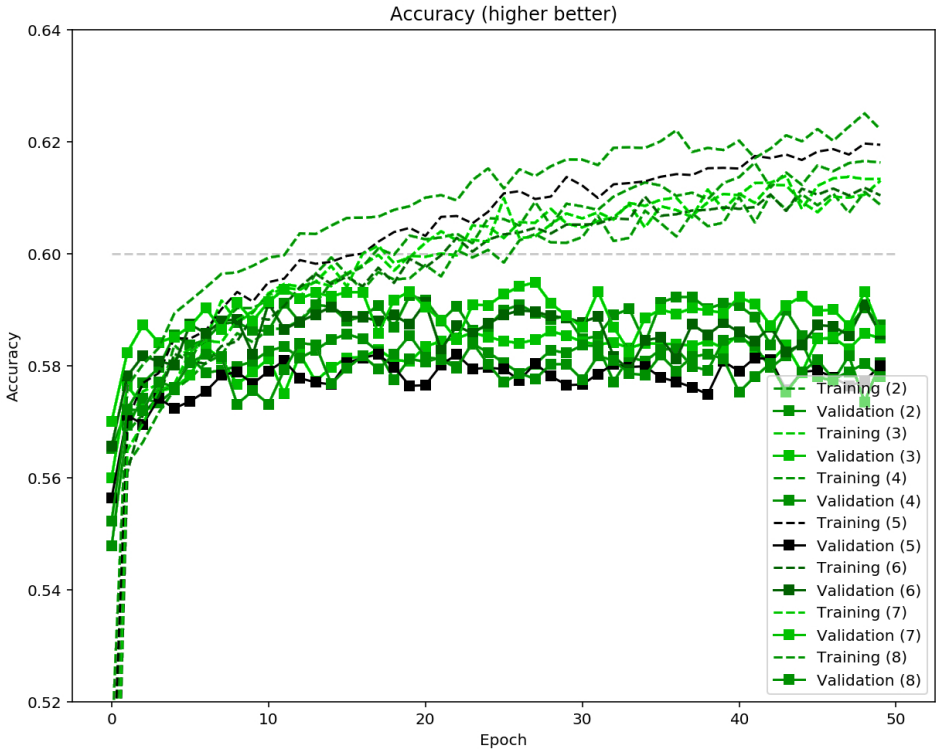
In this chapter we discuss the various approached used to tackle different kinds of datasets and the techniques used to handle the problems.

CHAPTER 5

Results & Discussion

5.1 Results





The plot of the training loss during the training of the twitter dataset on LSTM is shown in the graph below. The graph shows that the training loss decreases with increasing epochs and hence the learning rate is optimized.

The plot of the training accuracy during the training of the twitter dataset on LSTM is shown in the graph.

The graph shows that the training loss decreases with increasing epochs and hence the model is learning.

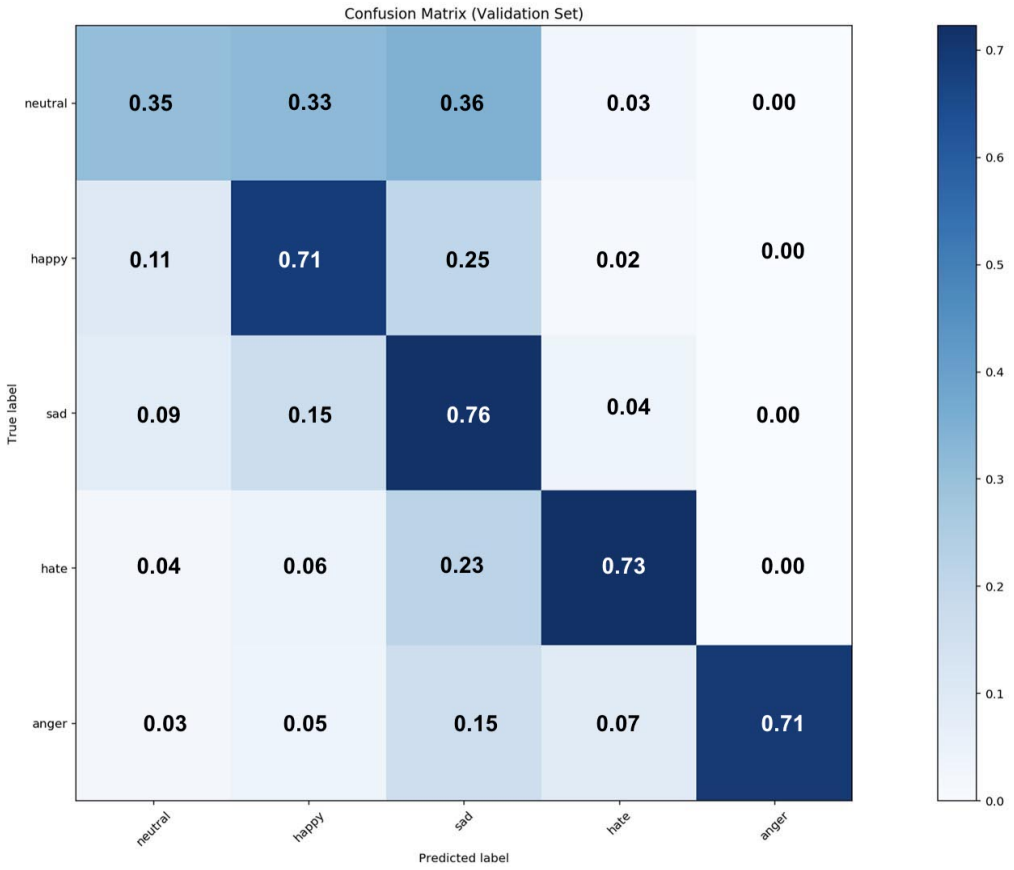
The maximum accuracy achieved by the various deep learning models deployed on various kinds of datasets gives the accuracy as stated in the table.

The maximum accuracy achieved by the LSTM model built for the advanced twitter dataset comprising of sarcastic remarks, multiple emotions and Singlish words attains an accuracy of 62.29%, competitive with the state of the art accuracy. The model is designed specifically for sarcastic remarks and multiple emotions make.

Table 5.1: Accuracy of various models

S.No.	Architecture	Accuracy
1	RNN 2 Emotions	86.01%
2	Faster RNN 2 Emotions	85.18%
3	CNN 2 Emotions	85.28%
4	CNN 5 Emotions	87.15%
5	LSTM 5 Emotions	62.29%

The covariance matrix for showcasing the results for the same dataset using LSTM is shown.Next, the delay estimation including stress variability was discussed in Chapter 5. For this estimation, the delay parameters k1-k3 are estimated from delay values at some stress points and then, it is predicted for further stress values.



5.2 Discussion

The models developed are able to achieve high accuracy comparable to the state of the art accuracy and even managed to surpass the same in the model being trained on the twitter advanced dataset comprising of sarcastic comments and Singlish language. Various descriptive plots shown such as covariance matrix and the learning curve justifies the fact that the model is not overfitting and learning.

CHAPTER 6

CONCLUSIONS

6.1 Conclusion

We managed to make a Recurrent Neural Network(RNN) which performed well on binary emotion datasets. We then reduced the training time to half. Thereafter we trained a Convolutional Neural Network(CNN) on a dataset with multiple emotions. After achieving high accuracy we laid hands on the advanced twitter dataset for which we tuned and adjusted the Recurrent Neural Network(RNN) which thereafter gave improved accuracy. Using techniques such as adaptive learning rate and dropouts helped us in making the model efficient in extracting maximum information from the datasets with minimal depth.

We accomplished all these results during the tenure of our project. We experienced how to handle real-time data and learned how to handle approaching deadlines.

6.2 Scope for Future Research

Feeling Detection is the most significant field of research in human-PC collaboration. Enough measure of work has been finished by specialists to distinguish feeling from facial and sound data though perceiving feelings from printed information is as yet a new and hot research zone, specifically for advanced datasets such as

- Sarcastic comments
- Slang words
- Incomplete sentences

We can improve and prepare our models to deal with a scope of conclusions. Tweets don't generally have a positive or negative opinion. On occasion, they may have no feeling for example impartial. Assumption can likewise have degrees like the sentence, This is great, is positive however the sentence, This is remarkable. is to some degree more positive than the first. We can subsequently characterize the feeling in extents, state from - 2 to +2.

During our pre-handling, we dispose of the greater part of the images like commas, full-stops, and shout mark. These images might be useful in doling out opinion to a sentence.

BIBLIOGRAPHY

[1] W. E.-H. H. H. Shaheen, Shadi and S. Elbassuoni, "Emotion recognition from text based on automatically generated rules," *Data Mining Workshop ICDMW*, vol. pp. 383-392, 2014.

[2] D. D. S. M. N. K. S. Tilakraj, Manasa M. and S. Narayan, "Ieee," in *Proc. 48th ACM/EDAC/IEEE Design Autom. Conf*, pp. 188–193, June 2014.

[3] A. A. S. S. R. B. Povoda, Lukas and M. K. Dutta, "Emotion recognition from helpdesk messages,"

[4] S. K. N. J. Jain, Vinay Kumar and P. Verma, "A novel approach to track public emotions related to epidemics in multilingual data," *International Conference and Youth School Information Technology and Nanotechnology*, vol. 101 No. 10, pp. 104503–1,Äì104503–22, 2016.

[5] N. P. Weerasooriya, Tharindu and S. R. Liyanage, "A method to extract essential keywords from a tweet using nlp tools," in *Advances in ICT for Emerging Regions (ICTer)*, 2016.

[6] S. N. Shivhare and S. Khethawat., "Emotion detection from text," Chapter-II, 2015.

[7] D. D. G. Prof. Hardik S. Jayswal and H. J. Gohil, "A review on a emotion detection and recognization from text using natural language processing,"

[8] K. R. L. Jin Wang, Liang-Chih Yu and X. Zhang, "Dimensional sentiment analysis using a regional cnn-lstm model," *Annual Meeting of the Association for Computational Linguistics*, 2016.

[9] X. W. Tao Chena Rui feng Xuab, Yulan Hec, "Improving sentiment analysis via sentence type classification using bilstm-crf and cnn," *Annual Meeting of the Association for Computational Linguistics*, vol. 36, pp. 1559–1564, 2017.

[10] S. H. Kashif khan and M. E. khan, "Emotion detection through text: Survey," *IEEE*, vol. 11, pp. 306–324, October 2012.

[11] S. K. V. S. Sunil B. Mane, Yashwant Sawant, "Real time sentiment analysis of twitter data using hadoop," *) International Journal of Computer Science and Information Technologies*, pp. 803–806, 2014.

[12] B. L. . E. B. . Y. C. . D. S. . G. Ch, "Scalable sentiment classification for big data analysis using naive bayes classifier," *IEEE J.*, vol. 25, no. 2, pp. 584–594, 2013.

APPENDIX A

A.I RNN for Binary Emotions

from torchtext import data

from torchtext import datasets

torch.manualseed(SEED)

torch.backends.cudnn.deterministic = True

from torchtext import datasets

traindata, testdata = datasets.IMDB.splits(TEXT, LABEL)

import random

traindata, validdata = traindata.split(randomstate = random.seed(SEED))

TEXT.buildvocab(traindata, maxsize = 25000, vectors = "glove.6B.100d", unkinit = torch.Tensor.normal)

LABEL.buildvocab(traindata)

device = torch.device('cuda' if torch.cuda.isavailable() else 'cpu')

trainiterator, validiterator, testiterator = data.BucketIterator.splits((traindata, validdata, testdata), batchsize = 64, sortwithinbatch = True, device = device)

import torch.nn as nn

class RNN(nn.Module):

```
def init(self, vocabsize, embeddingdim, hiddendim, outputdim, nlayers, bidirectional,
dropout, padidx):

super().init()

self.embedding = nn.Embedding(vocabsize, embeddingdim, paddingidx = padidx)

self.rnn = nn.LSTM(embeddingdim, hiddendim, numlayers=nlayers, bidirectional=bidirectional,
dropout=dropout)

self.fc = nn.Linear(hiddendim * 2, outputdim)

self.dropout = nn.Dropout(dropout)

def forward(self, text, textlengths):

embedded = self.dropout(self.embedding(text))

packedembedded = nn.utils.rnn.packpaddedsequence(embedded, textlengths)

packedoutput, (hidden, cell) = self.rnn(packedembedded)

output, outputlengths = nn.utils.rnn.padpackedsequence(packedoutput)

hidden = self.dropout(torch.cat((hidden[-2,:,:], hidden[-1,:,:]) dim = 1))

return self.fc(hidden.squeeze(0))

model = RNN(len(TEXT.vocab), 100, 256, 1, 2, True, 0.5, TEXT.vocab.stoi[TEXT.padtoken])

def countparameters(model):

return sum(p.numel() for p in model.parameters() if p.requiresgrad)

The model has 4,810,857 trainable parameters

pretrainedembeddings = TEXT.vocab.vectors

print(pretrainedembeddings.shape)

torch.Size([25002, 100])
```

UNKIDX = TEXT.vocab.stoi[TEXT.unktoken]

model.embedding.weight.data[UNKIDX] = torch.zeros(EMBEDDINGDIM)

model.embedding.weight.data[PADIDX] = torch.zeros(EMBEDDINGDIM)

import torch.optims as optims

optimizer = optims.Adam(model.parameters())

crit = nn.BCEWithLogitsLoss()

model = model.to(device) crit = crit.to(device)

def binaryaccuracy(preds, y):

roundedpredict = torch.round(torch.sigmoid(preds))

correct = (roundedpredict == y).float()

accuracy = correct.sum() / len(correct)

return accuracy

def train(model, iterator, optimizer, criterion):

epochlosses = 0

epochaccuracy = 0

model.train()

for batch in iterator:

optimizer.zero_grad()

text, textlnghs = batch.text

pred = model(text, textlnghs).squeeze(1)

loss = criterion(pred, batch.label)

acc = binaryaccuracy(pred, batch.label)

loss.backward()

optimizer.step()

epochlosses += loss.item()

epochaccuracy += acc.item()

return epochlosses / len(iterator), epochaccuracy / len(iterator)

def evaluate(model, iterator, criterion):

model.eval()

with torch.no_grad():

for batch in iterator:

text, textlnghs = batch.text

pred = model(text, textlnghs).squeeze(1)

loss = criterion(pred, batch.label)

acc = binaryaccuracy(pred, batch.label)

epochlosses += loss.item()

epochaccuracy += acc.item()

return epochlosses / len(iterator), epochaccuracy / len(iterator)

import time

def epochtime(startime, endingtime):

elapsedtime = endingtime - startime

elapsedminutes = int(elapsedtime / 60)

elapsedseconds = int(elapsedtime - (elapsedminutes * 60))

return elapsedminutes, elapsedseconds

bestvalidationloss = float('inf')

for epoch in range(5):

startime = time.time()

trainingloss, trainingacc = train(model, trainiterator, optimizer, criterion) validationloss, validationacc = evaluate(model, validiterator, criterion)

endtime = time.time()

epochmins, epochsecs = epochtime(starttime, endtime)

if validationloss < float('inf'): bestvalidationloss = float('inf') torch.save(model.statedict(), 'tut2-model.pt')

Epoch: 01 | Epoch Time: 0m 28s Train Loss: 0.641 | Train Acc: 62.54% Val. Loss: 0.520 | Val. Acc: 74.92% Epoch: 02 | Epoch Time: 0m 28s Train Loss: 0.559 | Train Acc: 71.77% Val. Loss: 0.735 | Val. Acc: 62.60% Epoch: 03 | Epoch Time: 0m 27s Train Loss: 0.479 | Train Acc: 78.00% Val. Loss: 0.403 | Val. Acc: 82.23% Epoch: 04 | Epoch Time: 0m 28s Train Loss: 0.381 | Train Acc: 83.92% Val. Loss: 0.343 | Val. Acc: 85.85%

model.loadstatedict(torch.load('tut2-model.pt'))

testloss, testacc = evaluate(model, testiterator, criterion)

print(f'Test Loss: testloss:.3f | Test Acc: testacc*100:.2f%')

Test Loss: 0.342 | Test Acc: 86.02%

import spacy nlp = spacy.load('en')

def predictsentiments(model, sentence): model.eval() tokenised = [tok.text for tok in nlp.tokenizer(sentence)] indexes = [TEXT.vocab.stoi[t] for t in tokenised] length = [len(indexes)] tensor = torch.LongTensor(indexed).to(device) tensors = tensor.unsqueeze(1) lengthtensors = torch.LongTensor(length) predictions = torch.sigmoid(model(tensors, lengthten-

sors)) return predictions.item()

predictsentiments(model, "This film is terrible")

0.019431069493293762

predictsentiment(model, "This film is great")

0.9544845819473267

A.II LSTM-CNN for Multiple Emotions

EMBEDDINGDIM = 5

embeddingmatrixns = np.random.random((2000, EMBEDDINGDIM))

for word, i in wordindex.items(): embeddingvector = embeddingsindex.get(word)

if embeddingvector is not None:

embeddingmatrixns[i] = embeddingvector

print("Completed!")

sequenceinput = Input(shape=(MAXSEQUENCELENGTH,), dtype='int32')

embeddinglayerfrozend = Embedding(len(wordindex) + 1, EMBEDDINGDIM,

weights=[embeddingmatrices], inputlength=MAXSEQUENCELENGTH, traineble=False)

embeddedsequencesfrozend = embeddinglayerfrozend(sequenceinput)

embeddinglayertrain = Embedding(len(wordindex) + 1, EMBEDDINGDIM,

weights=[embeddingmatrixns], inputlength=MAXSEQUENCELENGTH, trainable=True)

embeddedsequencestrain = embeddinglayertrain(sequenceinput)

llstm1first = Bidirectional(LSTM(6,returnsequences=True,dropout=0.3,

recurrentdropout=0.0))(embeddedsequencesfrozen)

```
llstm1top = Bidirectional(LSTM(6,returnsequences=True,dropout=0.3,
recurrentdropout=0.0))(embeddedsequencestrain)
llstm1 = Concatenate(axis=1)([llstm1first, llstm1top])
lconvolution2 = Conv1D(filters=24,kernelsize=2,activation='relu')(llstm1)
lconvolution2 = Dropout(0.3)(lconv2)
lconvolution3 = Conv1D(filters=24,kernelsize=3,activation='relu')(llstm1)
lconvolution3 = Dropout(0.3)(lconv3)
lconvolution5 = Conv1D(filters=24,kernelsize=5,activation='relu',)(llstm1)
lconvolution5 = Dropout(0.3)(lconv5)
lconvolution6 = Conv1D(filters=24,kernelsize=6,activation='relu',
kernelregularizer=regularizers.l2(0.0001))(llstm1)
lconvolution6 = Dropout(0.3)(lconv6)
lconvolutionolution8 = Conv1D(filters=24,kernelsize=8,activation='relu',
kernelregularizer=regularizers.l2(0.0001))(llstm1)
lconvolution8 = Dropout(0.3)(lconv8)
convolution1 = [lconvolution6,lconvolution5, lconvolution8,lconvolution2,lconvolution3]
llstmc = Concatenate(axis=1)(conv1)
lconvolution4f = Conv1D(filters=12,kernelsize=4,activation='relu',
kernelregularizer=regularizers.l2(0.0001))(embeddedsequencesfrozen)
lconvolution4f = Dropout(0.3)(lconv4f)
lconvolution4t = Conv1D(filters=12,kernelsize=4,activation='relu',
```

kernelregularizer=regularizers.l2(0.0001))(embeddedsequencestrain)

lconvolution4t = Dropout(0.3)(lconv4t)

lconvolution3f = Conv1D(filters=12,kernelsize=3,activation='relu',)(embeddedsequencesfrozen)

lconvolution3f = Dropout(0.3)(lconv3f)

lconvolution3t = Conv1D(filters=12,kernelsize=3,activation='relu',)(embeddedsequencestrain)

lconvolution3t = Dropout(0.3)(lconvolution3t)

lconvolution2f = Conv1D(filters=12,kernelsize=2,activation='relu')(embeddedsequencesfrozen)

lconvolution2f = Dropout(0.3)(lconvolution2f)

lconvolution2t = Conv1D(filters=12,kernelsize=2,activation='relu')(embeddedsequencestrain)

lconvolution2t = Dropout(0.3)(lconv2t)

convolution2 = [lconvolution4f, lconvolution4t,lconvolution3f, lconvolution3t, lconvolution2f, lconvolution2t]

lmerge2 = Concatenate(axis=1)(conv2)

lclstm = Bidirectional(LSTM(12,returnsequences=True,dropout=0.3,

recurrentdropout=0.0))(lmerge2)

lmerge = Concatenate(axis=1)([llstmc, lclstm])

lpooling = MaxPooling1D(4)(lmerge)

ldropping = Dropout(0.5)(lpooling)

lflatting = Flatten()(ldropping)

ldense = Dense(26, activation='relu')(lflatting)

prediction = Dense(5, activation='softmax')(ldense)

model = Model(sequenceinput, preds)

adadelta = optimizers.Adadelta(lr=0.9, rho=0.95, epsilon=None, decay=0.002)

lrmetric = getlrmetric(adadelta) model.compile(loss='categoricalcrossentropy', optimizer=adadelta, metrics=['acc'])

tensorboard = callbacks.TensorBoard(logdir='./logs',histogramfreq=0, batchsize=16, writegrads=True , writegraph=True)

modelcheckpoints = callbacks.ModelCheckpoint("checkpoint-valloss:.3f.h5", monitor='valloss', verbose=0, savebestonly=True, saveweightsonly=False, mode='auto', period=0)

lrscheduler = callbacks.LearningRateScheduler(initialboost)

model = keras.models.loadmodel("checkpoint-0.81.h5")

modellog = model.fit(xtrain, ytrain, validationdata=(xval, yval), epochs=200, batchsize=128, callbacks=[tensorboard, modelcheckpoints])

pandas.DataFrame(modellog.history).tocsv("history-balance.csv")

from keras.models import loadmodel

from sklearn.metrics import classificationreport, confusionmatrix

import matplotlib.pyplot as plt

import numpy as np %config InlineBackend.figureformat = 'retina'

import itertools, pickle

with open('tokenizer.pickle', 'rb') as handle: tokeniser = pickle.load(handle)

classes = ["neutral", "happy", "sad", "hate","anger"]

modeltest = loadmodel('bestweights.h5')

Ytesting = np.argmax(yvalidation, axis=1) Convert one-hot to index

yprediction = modeltest.predict(xvalidation)

ypredclass = np.argmax(yprediction,axis=1)

cnfmatrix = confusionmatrix(Ytestting, ypredclass)

print(classificationreport(Ytestting, ypredclass, targetnames=classes))

def plotconfusionmatrix(cm, labels, normalize=True, title='Confusion Matrix (Validation Set)', cmap=plt.cm.Blues):

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

else:

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tickmarks = np.arange(len(labels))

plt.xticks(tickmarks, labels, rotation=45)

plt.yticks(tickmarks, labels)

fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, format(cm[i, j], fmt), horizontalalignment="center", color="white" if cm[i, j] > thresh else "black")

plt.tightlayout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.figure(figsize=(20,10))

plotconfusionmatrix(cnfmatrix, labels=classes)

precision = truepos / (truepos + falsepos)

recall = truepos / (truepos + falseneg)

text = [

"I salute you for the bravery and sacrifice! A true hero indeed.",

"I am sorry but I trust HRW I damned sight more than the PAP and it's cronies! Off course the PAP will say that they (HRW) made things up...despite of the fact that SG is a dictator state!?",

"PAP are taking the piss again!",

"Thought he sold his kidney to buy it; Instead, he bought a kidney then bought the car Filthy rich This is why we need communism",

"Somebody needs to water Tharman's head, hair needs to be grown there",

"what a nuisance fk. a proper clean and flat footpath„now obstructed by sharedbikes..! which idiotic MP allowed this to happen?",

"What baby bonus scheme ??? To grow up a kid in Singapore you think is easy now bo ??? Both parent need to work to grow up a kid until 21 , you think tats easy bo ??? Think la"]

sequencestest = tokeniser.textstosequences(text)

dataintt = padsequences(sequencestest, padding='pre', maxlen=(MAXSEQUENCELENGTH-5))

datatest = padsequences(dataintt, padding='post', maxlen=(MAXSEQUENCELENGTH))

yprob = model.predict(datatest)

for n, prediction in enumerate(yprob): pred = yprob.argmax(axis=-1)[n]

print(text[n],":",classes[pred])

I salute you for the bravery and sacrifice! A true hero indeed. Prediction: happy

I am sorry but I trust HRW I damned sight more than the PAP and it's cronies! Off

course the PAP will say that they (HRW) made things up...despite of the fact that SG is a dictator state!? Prediction: sad

PAP are taking the piss again! Prediction: sad

Thought he sold his kidney to buy it; Instead, he bought a kidney then bought the car Filthy rich This is why we need communism Prediction: hate

Somebody needs to water Tharman's head, hair needs to be grown there Prediction: sad

what a nuisance fk. a proper clean and flat footpath„now obstructed by sharedbikes..! which idiotic MP allowed this to happen? Prediction: sad

What baby bonus scheme ??? To grow up a kid in Singapore you think is easy now bo ??? Both parent need to work to grow up a kid until 21 , you think tats easy bo ??? Think la Prediction: sad

np.array(sequencestest[0])

array([1, 6097, 5, 12, 3, 6, 4841, 4, 515, 1476, 1238])