HOTEL REVIEWS BASED SENTIMENTAL ANALYSIS

A Project Report

Submitted in Partial Fulfillment of the Requirements for the Award of the Degree of

Bachelor of Technology

In

Information Technology

Submitted by

Shubham Kumar (1607004)
Amid Ahmed (1607036)

Under the Joint supervision of:

Dr. Kumar Abhishek (Asst. Professor)

Department of Computer Science and Engineering

National Institute of Technology Patna

Department of Computer Science and Engineering

Jan-Jun, (2019-2020)

NATIONAL INSTITUTE OF TECHNOLOGY PATNA
(An Institute under Ministry Of HRD, Govt. of India)
ASHOK RAJPATH, PATNA-800005 (BIHAR)

CERTIFICATE

This is to certify that **Shubham Kumar (1607004), Amid Ahmed (1607036)** have worked on **"Hotel Review Based Sentimental Analysis"** as their Major Project (8IT195) under my supervision and as a part of his Internship.

I hereby recommend that this project to be accepted as per the requirements of evaluation and for the award of B.Tech Degree.

Dr. Kumar Abhishek
Assistant Professor
Computer Science & Engineering
National Institute of Technology, Patna

Dr. Jyoti Prakash Singh
Head of Department
Computer Science & Engineering
National Institute of Technology, Patna

राष्ट्रीय प्रौद्योगिकी संस्थान पटना
NATIONAL INSTITUTE OF TECHNOLOGY PATNA

Declaration

We hereby declare that this project work for Major Project (8IT195) entitled **"Hotel Review Based Sentimental Analysis"** is an authentic record of my own work carried out at **National Institute of Technology, Patna** under the joint supervision of **Dr. Kumar Abhishek**, (Assistant Professor, Department of Computer Science and Engineering, National Institute of Technology Patna).

No part of this project has been presented and will not be presented to any other University/Institute for a similar or any other Degree award.

S.No.	**Name**	**Roll No.**
1.	SHUBHAM KUMAR	1607004
2.	AMID AHMED	1607036

Date:
Place: Patna

Acknowledgement

I hereby take the privilege to express my gratitude to all the people who are directly or indirectly involved in the execution of this work, without whom this project would not have been a success.

I would like to express my deepest gratitude, respect and obligation to our project supervisor, **Dr. Kumar Abhishek**, (Assistant Professor, Department of Computer Science and Engineering, National Institute of Technology Patna) for the valuable suggestions and directions throughout the development of the project.

I further express my gratitude to the Head of the Computer Science and Engineering Department, **Dr. Jyoti Prakash Singh** for being a constant source of inspiration.

Finally, I would like to thank my friends and family for constantly supporting me in this endeavor. As a special mention I would like to thank my team members for being with me on this project throughout and understanding its nuances and bringing it to its conclusion.

ShubhamKumar (1607004)

Amid Ahmed (1607036)

Date:
Place: Patna

Abstract

Sentiment analysis or opinion mining is the study in which it analyzes people's opinions, sentiments, evaluations, attitudes, and emotions from natural written language. It is an active research area in natural language processing and in the field of data mining. It has grown widely due to its importance to business and society. The sentiment analysis is used widely in the field of social media and other reviewing systems.

Travel planning and hotel bookings have become significant commercial applications. In recent years, there has been a rapid growth in online review sites and discussion forums where the critical characteristics of a customer review are drawn from their overall opinion/sentiments. Customer reviews play a significant role in a hotel's persona which directly affects its valuation. This research work is intended to address the problem of analyzing the inundation of opinions and reviews of hotel services publicly available over the Web. Availability of large datasets containing such texts has allowed us to automate the task of sentiment profiling and opinion mining. In this study, we have collected Europe Hotels reviews dataset, from which we have collected both Positive and Negative reviews, and after pre-processing of collected raw text reviews using different preprocessing techniques, various features are extracted, and then using different Deep Learning models the results are compared to find best performance measures such as accuracy, F-measure, precision, and recall.

Sentiment analysis is the procedure by which information is extracted from the opinions, appraisals and emotions of people in regards to entities, events and their attributes. In decision making, the opinions of others have a significant effect on customers ease, making choices with regards to online shopping, choosing events, products, entities. The approaches of text sentiment analysis typically work at a particular level like phrase, sentence or document level. This report aims at analyzing a solution for the sentiment analysis using different preprocessing techniques and deep learning Models to find out comparative study among those models.

CONTENTS

Introduction

Sentiment analysis is the process of using natural language processing, text analysis, and statistics to analyze customer sentiment. The best businesses understand the sentiment of their customers—what people are saying, how they're saying it, and what they mean. Customer sentiment can be found in tweets, comments, reviews, or other places where people mention your brand. Sentiment Analysis is the domain of understanding these emotions with software, and it's a must-understand for developers and business leaders in a modern workplace.

As with many other fields, advances in deep learning have brought sentiment analysis into the foreground of cutting-edge algorithms. Today we use natural language processing, statistics, and text analysis to extract, and identify the sentiment of words into positive, negative, or neutral categories.

Sentiment analysis refers to the use of natural language processing, text analysis and computational linguistics to identify and extract subjective information in source materials. Generally speaking, sentiment analysis aims to determine the attitude of a speaker or a writer with respect to some topic or the overall contextual polarity of a document. The attitude may be his or her judgment or evaluation affective state, or the intended emotional communication. Sentiment analysis is the process of detecting a piece of writing for positive, negative, or neutral feelings bound to it. Humans have the innate ability to determine sentiment; however, this process is time consuming, inconsistent, and costly in a business context It's just not realistic to have people individually read tens of thousands of user customer reviews and score them for sentiment.

What is a Review?

A review is an evaluation procedure of a service, a product, a publication or any other company. In addition to critical evaluation, the review's author assigns the work a rating to indicate its relative merit. More loosely, an author may review current events or trends in the news. A compilation of reviews itself may be called a review.

A user review refers to a review written by a user for a product or a service based on his/her experience as a user of the reviewed hotel. Popular sources for consumer reviews are e-commerce & social media sites such as Amazon.com, Trip Advisor etc., Usually, customer reviews are in the form that is of several lines of texts along with a numerical rating. This helps the user to select a respective hotel. A customer review of a service usually comment on how well the service is useful for them and how well they meet the measures up to expectations based on the specification given by the hotel. It talks about anything such as the cleanliness, liesures, if any, and value for money. Customer review is also known as the user generated content, is different from that of the marketer-generated content in its evaluation from user point of view. It includes comparative evaluations against competing hotels. Perceptions are true as well as emotional in nature. Customer review of hotels usually comments on service

experienced, trustworthiness of the hotel. It comments on factors such as timeliness of foods, the service provided, cleanliness and so on.

1.1 Objective

- To train a model on reviews based sentimental analysis by which we can help the customers to get best hotels.
- To use different pre-processing techniques by which we can categorize our dataset differently for model training.
- To make comparative study of different models for sentiment analysis to find best model among them using accuracy.

1.2 Proposed Approach and Methods to be Employed

Sentiment Analysis or Opinion Mining is a study that attempts to identify and analyze emotions and subjective information from text since early 2001, the advancement of internet technology and machine learning techniques in information retrieval make Sentiment Analysis becomes popular among researchers. Besides, the emergent of social networking and blogs as a communication medium also contributes to the development of research in this area Sentiment analysis or mining refers to the application of Natural Language Processing, Computational Linguistics, and Text Analytics to identify and extract subjective information in source materials. Sentiment mining extracts attitude of a writer in a document includes writer's judgement and evaluation towards the discussed issue.

Sentiment analysis allows us to identify the emotional state of the writer during writing, and the intended emotional effect that the author wishes to give to the reader. In recent years, sentiment analysis becomes a hotspot in numerous research fields, including Deep Learning, natural language processing (NLP), data mining (DM). This is due to the increasing of subjective texts appearing on the internet. Deep Learning is commonly used to classify sentiment from text. This technique involves with model such as CNN (Convolutional Neural Networks), RNN(Recurrent Neural Networks), HAN(Hierarchical Attention Network) and RMDL (Random Multimodel for Deep Learning) .The processing techniques used for text classification are glove, Word2Vec, Fasttext and BERT(Bidirectional Encoder Representations from Transformers). The most commonly used in sentiment mining were taken from blog, twitter and web review which focusing on sentences that expressed sentiment directly. The main aim of this project is to develop a model that helps to find best hotels based on accuracy score of the different hotel reviews.

2 Literature Survey

2.1 Pre-processing Techniques

- **Word2Vec**- It is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located close to one another in the space.

Word2Vec is a method to construct such an embedding. It can be obtained using two methods (both involving Neural Networks): Skip Gram and Common Bag Of Words (CBOW)

CBOW Model: This method takes the context of each word as the input and tries to predict the word corresponding to the context. Consider our example: *Have a great day.*

Let the input to the Neural Network be the word, *great.* Notice that here we are trying to predict a target word (*d*ay*)* using a single context input word *great.* More specifically, we use the one hot encoding of the input word and measure the output error compared to one hot encoding of the target word (*d*ay). In the process of predicting the target word, we learn the vector representation of the target word.

Let us look deeper into the actual architecture.

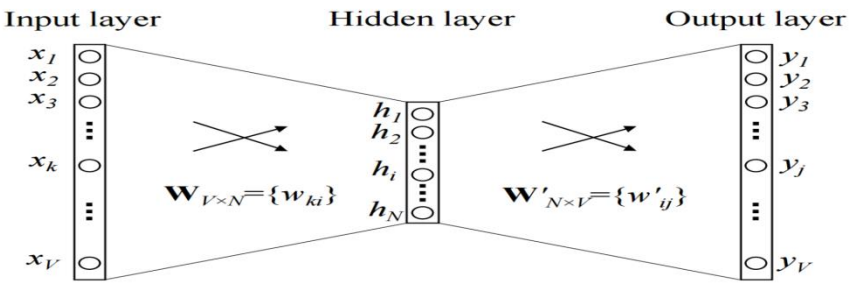


Figure 1: A simple CBOW model with only one word in the context

The input or the context word is a one hot encoded vector of size V. The hidden layer contains N neurons and the output is again a V length vector with the elements being the softmax values.

Let's get the terms in the picture right:
- *Wvn is the weight matrix that maps the input x to the hidden layer (V*N dimensional matrix)*
*-W`nv is the weight matrix that maps the hidden layer outputs to the final output layer (N*V dimensional matrix)*

I won't get into the mathematics. We'll just get an idea of what's going on.

The hidden layer neurons just copy the weighted sum of inputs to the next layer. There is no activation like sigmoid, tanh or ReLU. The only non-linearity is the softmax calculations in the output layer.

But, the above model used a single context word to predict the target. We can use multiple context words to do the same.

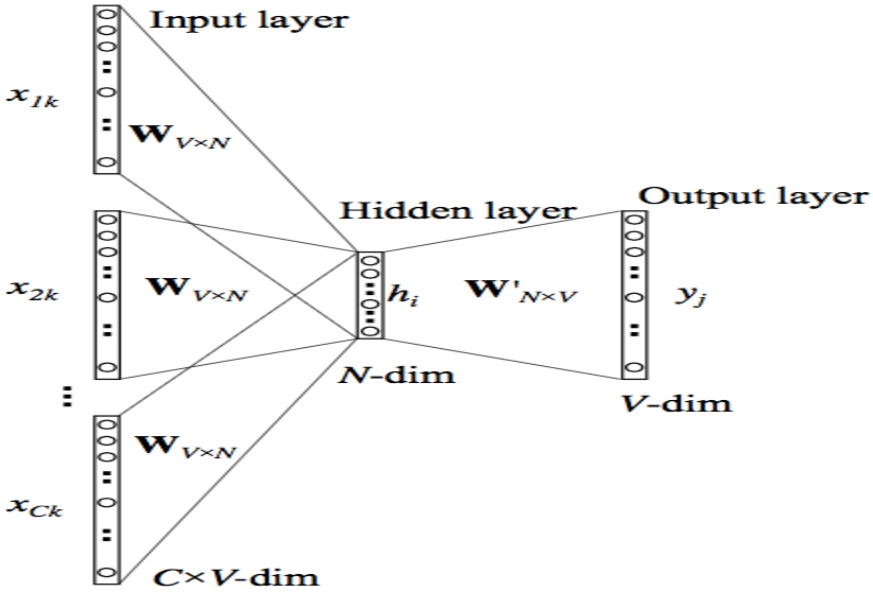


Figure:2- A CBOW model with multiple Words in the context

The above model takes C context words. When *Wvn* is used to calculate hidden layer inputs, we take an average over all these C context word inputs.

So, we have seen how word representations are generated using the context words. But there's one more way we can do the same. We can use the target word (whose representation we want to generate) to predict the context and in the process, we produce the representations. Another variant, called Skip Gram model does this.

Skip-Gram model:

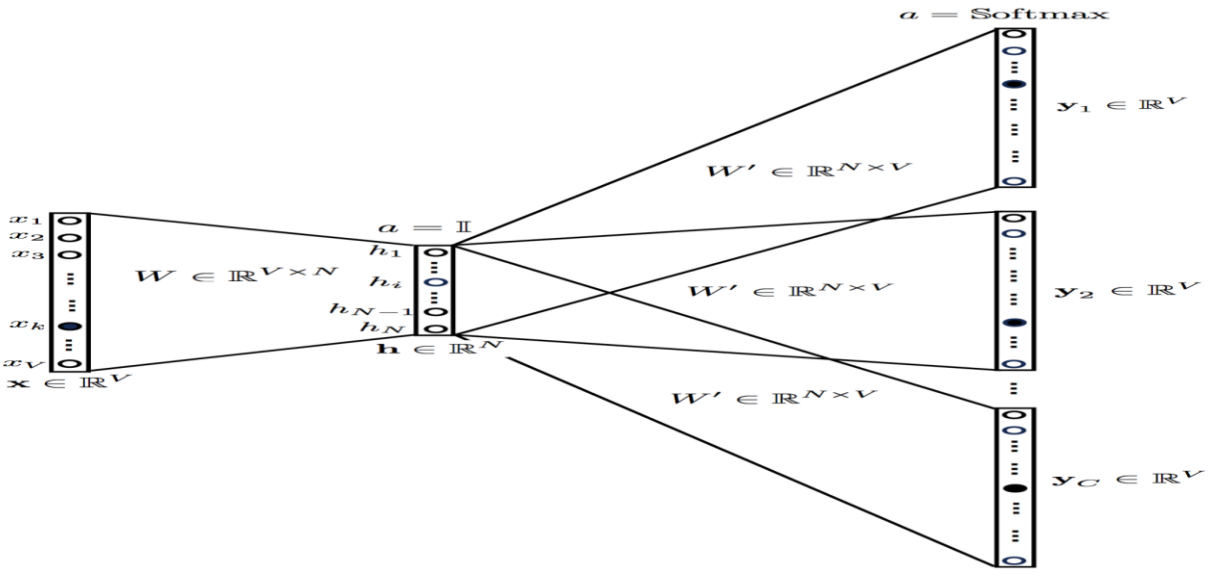


Figure 3: Image of a Skip Gram Model

This looks like multiple-context CBOW model just got flipped. To some extent that is true.

We input the target word into the network. The model outputs C probability distributions. What does this mean?

For each context position, we get C probability distributions of V probabilities, one for each word.

In both the cases, the network uses back-propagation to learn.

- **Fasttext-** fastText is a library for learning of word embeddings and text classification created by Facebook's AI Research (FAIR) lab. The model allows to create an unsupervised learning or supervised learning algorithm for obtaining vector representations for words. Facebook makes available pretrained models for 294 languages. fastText uses a neural network for word embedding.

fastText as a library for efficient learning of word representations and sentence classification. It is written in C++ and supports multiprocessing during training. FastText allows you to train supervised and unsupervised representations of words and sentences. These representations (embeddings) can be used for numerous applications from data compression, as features into additional models, for candidate selection, or as initializers for transfer learning.

FastText supports training continuous bag of words (CBOW) or Skip-gram models using negative sampling, softmax or hierarchical softmax loss functions. I have

primarily used fastText for training semantic embeddings for a corpus of size in the order of tens millions, and am happy with how it has performed and scaled for this task. I had a hard time finding documentation beyond the documentation for getting started, so in this post I am going to walk you through the internals of fastText and how it works. An understanding of how the word2vec models work is expected.

Running fastText

We can train a Skip-gram model via fastText with the following command:

$ fasttext skipgram -input data.txt -output model

where data.txt is the input data which can just be a sequence of text, and the output model gets saved under model.bin and vector representations for the input terms are saved under model.vec.

Representation

FastText is able to achieve really good performance for word representations and sentence classification, specially in the case of rare words by making use of character level information.

Each word is represented as a bag of character n-grams in addition to the word itself, so for example, for the word matter, with n = 3, the fastText representations for the character n-grams is <ma, mat, att, tte, ter, er>. < and > are added as boundary symbols to distinguish the ngram of a word from a word itself, so for example, if the word mat is part of the vocabulary, it is represented as <mat>. This helps preserve the meaning of shorter words that may show up as n-grams of other words. Inherently, this also allows you to capture meaning for suffixes/prefixes.

The length of n-grams you use can be controlled by the -minn and -maxn flags for minimum and maximum number of characters to use respectively. These control the range of values to get n-grams for. The model is considered to be a bag of words model because aside of the sliding window of n-gram selection, there is no internal structure of a word that is taken into account for featurization, i.e as long as the characters fall under the window, the order of the character n-grams does not matter. You can also turn n-gram embeddings completely off as well by setting them both to 0. This can be useful when the ‘words’ in your model aren’t words for a particular language, and character level n-grams would not make sense. The most common use case is when you’re putting in ids as your words. During the model update, fastText learns weights for each of the n-grams as well as the entire word token.

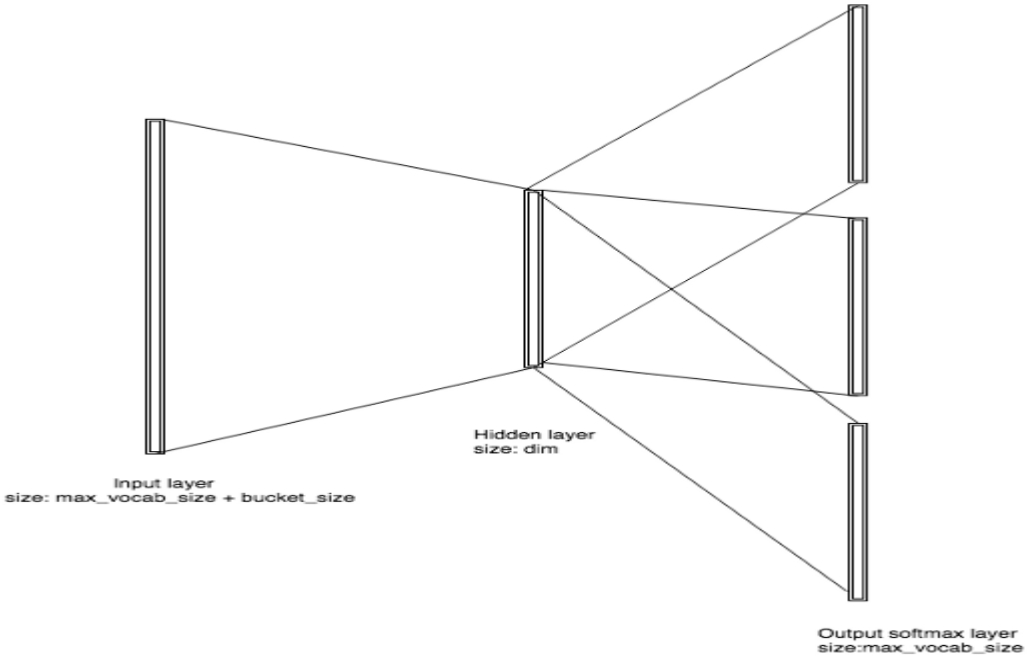


Figure 4: Fasttext model in Update Phase

The model input weights, hidden layer weights along with arguments passed in are saved in the .bin format and the -saveOutput flag controls whether a .vec file is also outputted which contains vectors for the hidden layer in the word2vec file format.

Glove-(Global Vectors)

GloVe is another word embedding method. But it uses a different mechanism and equations to create the embedding matrix. To study GloVe, let's define the following terms first.

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

$X_i = \sum_k X_{ik}$

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

And the ratio of co-occurrence probabilities as:

$w \in \mathbb{R}^d$ are word vectors — probe word

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

co-relations between the word wi and wj — co-occurrence probabilities for the word wj and wk

This ratio gives us some insight on the co-relation of the probe word wk with the word w_i and w_j.

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

Given a probe word, the ratio can be small, large or equal to 1 depends on their correlations. For example, if the ratio is large, the probe word is related to w_i but not w_j. This ratio gives us hints on the relations between three different words. Intuitively, this is somewhere between a bi-gram and a 3-gram.

Now, we want to develop a model for F given some desirable behavior we want for the embedding vector w. As discussed before, linearity is important in the word embedding concept. So if a system is trained on this principle, we should expect that F can be reformulated as:

$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{w_i^T \tilde{w}_k}{w_j^T \tilde{w}_k} = \frac{P_{ik}}{P_{jk}}$$

($w_i^T \tilde{w}_k$ relate to (high probability if they are similar))

where we just need to compute the difference and the similarity of word embedding for the parameters in F.

In addition, their relation is symmetrical. (a.k.a. relation(a, b) = relation(b, a)). To enforce such symmetry, we can have

$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)}$$

Intuitively, we are maintaining the linear relationship among all these embedding vectors.

To fulfill this relation, $F(x)$ would be an exponential function, i.e. $F(x) = \exp(x)$. Combine the last two equations, we get

$$F(w_i^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i}$$

Since $F(x) = \exp(x)$,

$$w_i^T \tilde{w}_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i)$$
$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})$$

($\log(X_{ik})$: co-occurrence count for word wi and wk)

We can absorb $\log(X_i)$ as a constant bias term since it is invariant of k. But to maintain the symmetrical requirement between i and k, we will split it into two bias terms above. This w and b form the embedding matrix. Therefore, the dot product of two embedding matrices predicts the log co-occurrence count.

Intuition

Let's understand the concept through matrix factorization in a recommender system. The vertical axis below represents different users and the horizontal axis represents different movies. Each entry shows the movie rating a user gives.

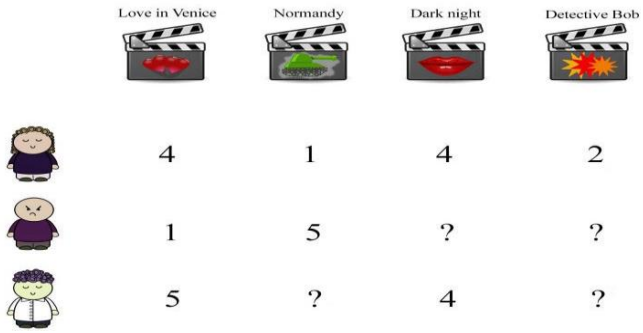


Figure 5: Image of a Matrix Factorization Problem

This can be solved as a matrix factorization problem. We want to discover the hidden factors for the users and movies. This factor describes what a user likes or what the hidden features (like the genre) a movie will be. If their factors match, the movie rating will be high. For example, if a user likes romantic and old movies, they will match well with the movie "When Harry Met Sally" (a romantic movie in the 80s). The vector representations for the user and the movie should produce high value for their dot product.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & ? & ? & \dots & r_{1n} \\ r_{21} & r_{22} & ? & r_{24} & ? & \dots & ? \\ & & \vdots & \vdots & \ddots & \vdots & \vdots \\ r_{m1} & ? & r_{m3} & ? & r_{m5} & \dots & r_{mn} \end{bmatrix} \approx \begin{bmatrix} z_{11} & z_{12} & z_{13} & \dots & z_{1k} \\ z_{21} & z_{22} & z_{23} & \dots & z_{2k} \\ & & \vdots & \ddots & \vdots \\ z_{m1} & z_{m2} & z_{m3} & \dots & z_{mk} \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1k} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2k} \\ & & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & w_{n3} & \dots & w_{nk} \end{bmatrix}^T$$

$$J(W, Z) = \sum_i \sum_j (W_j^T z_i - r_{ij})^2 + \frac{\lambda_1}{2} \|W\|_f^2 + \frac{\lambda_2}{2} \|Z\|_f^2$$

Therefore the rating matrix holding all users and movies can be approximated as the multiplication of the users' hidden features and the movies' hidden features (matrix *Z* holding the hidden factors of all users and *w* hold the hidden factors for all movies).

In GloVe, we measure the similarity of the hidden factors between words to predict their co-occurrence count. Viewed from this perspective, we do not predict the co-occurrence words only. We want to create vector representations that can predict their co-occurrence counts in the corpse also.

$$\underbrace{w_i^T \tilde{w}_k}_{\text{measures the similarity of the hidden factors between both words to predict co-occurrence count}} + b_i + \tilde{b}_k = \log(X_{ik})$$

Cost function

Next, we will define the cost function. We will use the Mean Square Error to calculate the error in the ground truth and the predicted co-occurrence counts. But since word pair have different occurrence frequency in the corpus, we need a weight to readjust the cost for each word pair. This is the function *f* below. When the co-occurrence count is higher or equal a threshold, say 100, the weight will be 1. Otherwise, the weight will be smaller, subject to the co-occurrence count. Here is the objective function in training the GloVe model.

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

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

(100 over $x_{\max}$, 3/4 over α)

- **BERT-** BERT (Bidirectional Encoder Representations from Transformers) is a recent paper published by researchers at Google AI Language. It has caused a stir in the Machine Learning community by presenting state-of-the-art results in a wide variety of NLP tasks, including Question Answering (SQuAD v1.1), Natural Language Inference (MNLI), and others.

BERT's key technical innovation is applying the bidirectional training of Transformer, a popular attention model, to language modeling. This is in contrast to previous efforts which looked at a text sequence either from left to right or combined left-to-right and right-to-left training. The paper's results show that a language model which is bi-directionally trained can have a deeper sense of language context and flow than single-direction language models. In the paper, the researchers detail a novel technique named Masked LM (MLM) which allows bidirectional training in models in which it was previously impossible.

Background

In the field of computer vision, researchers have repeatedly shown the value of transfer learning — pre-training a neural network model on a known task, for instance ImageNet, and then performing fine-tuning — using the trained neural network as the basis of a new purpose-specific model. In recent years, researchers have been showing that a similar technique can be useful in many natural language tasks.

A different approach, which is also popular in NLP tasks and exemplified in the recent ELMo paper, is feature-based training. In this approach, a pre-trained neural network produces word embeddings which are then used as features in NLP models.

How BERT works

BERT makes use of Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. In its vanilla form, Transformer

includes two separate mechanisms — an encoder that reads the text input and a decoder that produces a prediction for the task. Since BERT's goal is to generate a language model, only the encoder mechanism is necessary. The detailed workings of Transformer are described in a paper by Google.

As opposed to directional models, which read the text input sequentially (left-to-right or right-to-left), the Transformer encoder reads the entire sequence of words at once. Therefore it is considered bidirectional, though it would be more accurate to say that it's non-directional. This characteristic allows the model to learn the context of a word based on all of its surroundings (left and right of the word).

The chart below is a high-level description of the Transformer encoder. The input is a sequence of tokens, which are first embedded into vectors and then processed in the neural network. The output is a sequence of vectors of size H, in which each vector corresponds to an input token with the same index.

When training language models, there is a challenge of defining a prediction goal. Many models predict the next word in a sequence (e.g. "The child came home from ___"), a directional approach which inherently limits context learning. To overcome this challenge, BERT uses two training strategies:

Masked LM (MLM)

Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token. The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence. In technical terms, the prediction of the output words requires:

Adding a classification layer on top of the encoder output.

Multiplying the output vectors by the embedding matrix, transforming them into the vocabulary dimension.

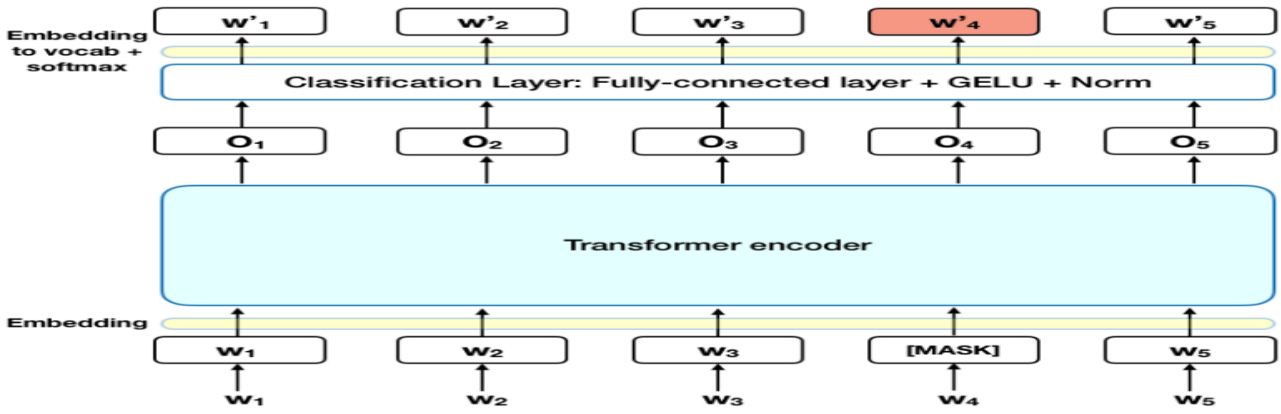


Figure 6: Calculating the probability of each word in the vocabulary with softmax.

The BERT loss function takes into consideration only the prediction of the masked values and ignores the prediction of the non-masked words. As a consequence, the model converges slower than directional models, a characteristic which is offset by its increased context awareness.

Next Sentence Prediction (NSP)

In the BERT training process, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document. During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50% a random sentence from the corpus is chosen as the second sentence. The assumption is that the random sentence will be disconnected from the first sentence.

To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:

A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.

A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of 2.

A positional embedding is added to each token to indicate its position in the sequence. The concept and implementation of positional embedding are presented in the Transformer paper.

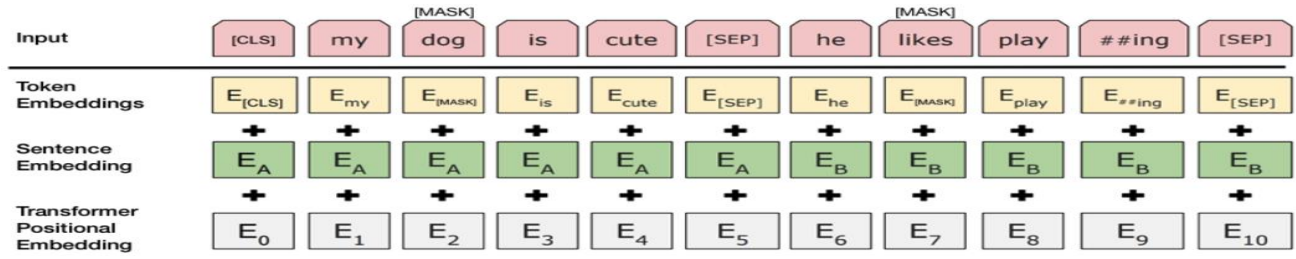


Figure 7: Image of Next Sentence Prediction Process

To predict if the second sentence is indeed connected to the first, the following steps are performed:

The entire input sequence goes through the Transformer model.

The output of the [CLS] token is transformed into a 2×1 shaped vector, using a simple classification layer (learned matrices of weights and biases).

Calculating the probability of IsNextSequence with softmax.

When training the BERT model, Masked LM and Next Sentence Prediction are trained together, with the goal of minimizing the combined loss function of the two strategies.

Models

CNN- A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

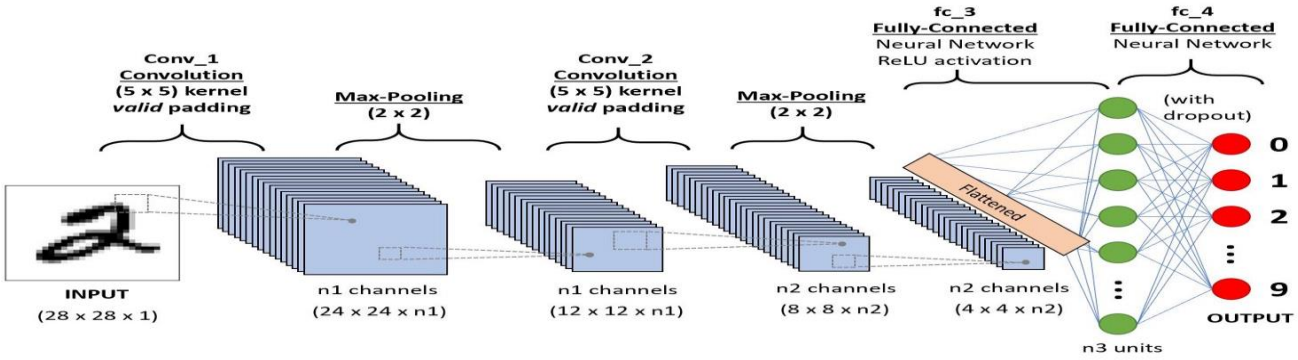


Figure 8: A Fully Connected CNN (Convolutional Neural Network)

Why ConvNets over Feed-Forward Neural Nets?

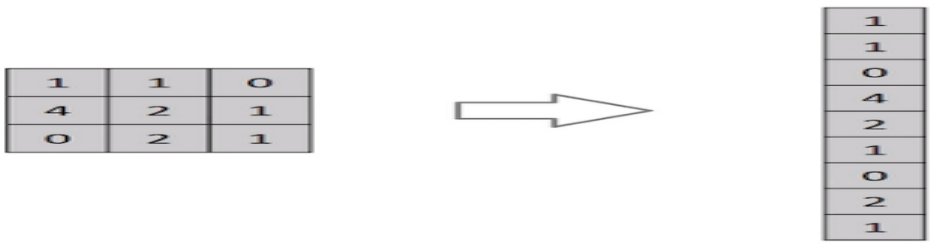


Figure 9: Flattening of a 3x3 image matrix into a 9x1 vector

An image is nothing but a matrix of pixel values, right? So why not just flatten the image (e.g. 3x3 image matrix into a 9x1 vector) and feed it to a Multi-Level Perceptron for classification purposes.

In cases of extremely basic binary images, the method might show an average precision score while performing prediction of classes but would have little to no accuracy when it comes to complex images having pixel dependencies throughout.

A ConvNet is able to **successfully capture the Spatial and Temporal dependencies** in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

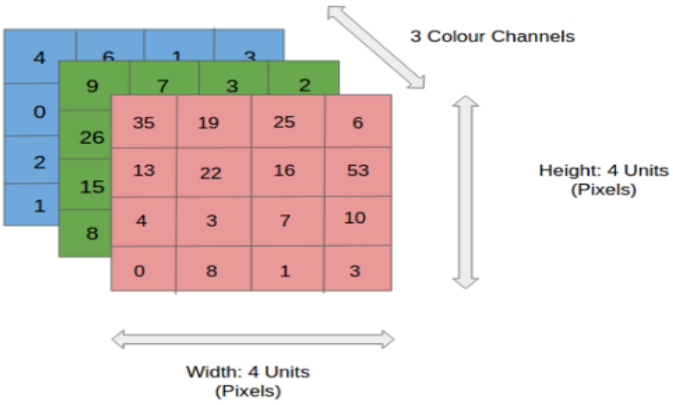


Figure 10: Image showing RGB color code for different blocks

In the figure, we have an RGB image which has been separated by its three color planes — Red, Green, and Blue. There are a number of such color spaces in which images exist — Grayscale, RGB, HSV, CMYK, etc.

You can imagine how computationally intensive things would get once the images reach dimensions, say 8K (7680×4320). The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. This is important when we are to design an architecture which is not only good at learning features but also is scalable to massive datasets.

Convolution Layer — The Kernel

Figure 11: A 5*5*1 and 3*3*1 kernel

Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB)

In the above demonstration, the green section resembles our **5x5x1 input image, I**. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the **Kernel/Filter, K**, represented in the color yellow. We have selected **K as a 3x3x1 matrix.**

Kernel/Filter, K = 1 0 1
0 1 0
1 0 1

The Kernel shifts 9 times because of **Stride Length = 1 (Non-Strided)**, every time performing a **matrix multiplication operation between K and the portion P of the image** over which the kernel is hovering.

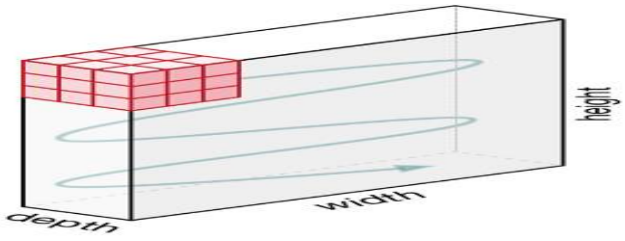


Figure 12: Movement of Kernel

Movement of the Kernel

The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.

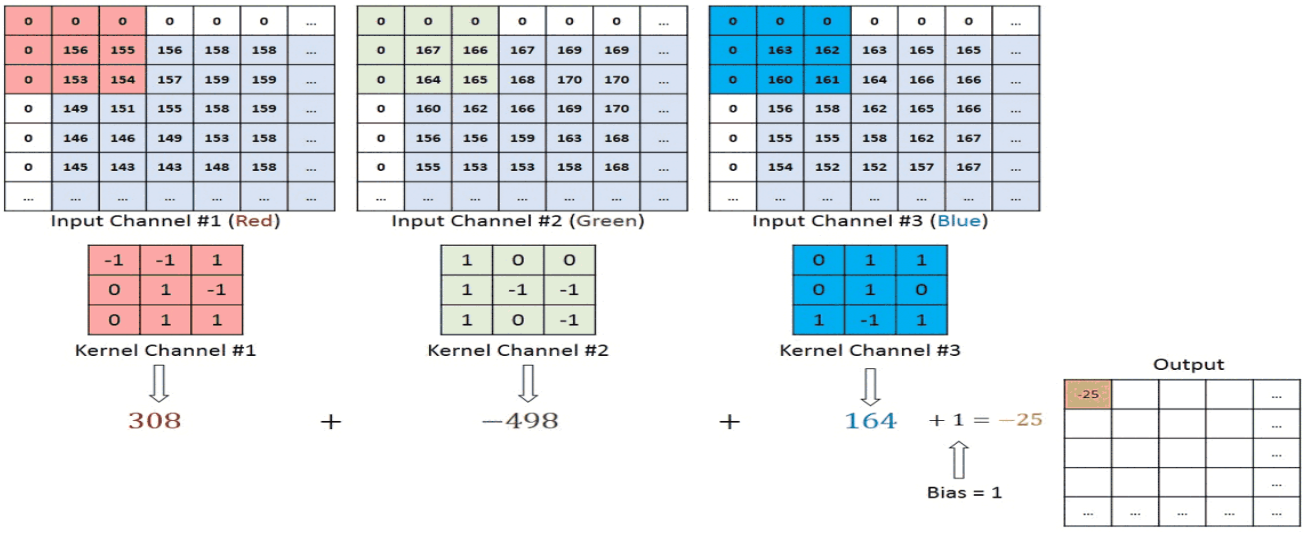


Figure 13: Convolution operation on a MxNx3 image matrix with a 3x3x3 Kernel

In the case of images with multiple channels (e.g. RGB), the Kernel has the same depth as that of the input image. Matrix Multiplication is performed between Kn and In stack

([K1, I1]; [K2, I2]; [K3, I3]) and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output.

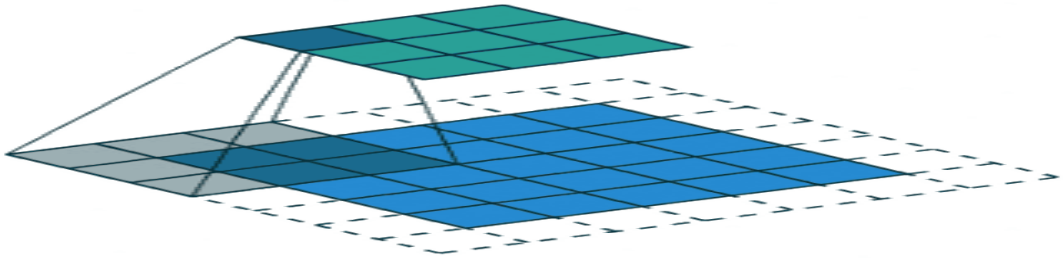


Figure 14: One Depth channel Convoluted Feature Output

Convolution Operation with Stride Length = 2

The objective of the Convolution Operation is to **extract the high-level features** such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.

There are two types of results to the operation — one in which the convolved feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either increased or remains the same. This is done by applying **Valid Padding** in case of the former, or **Same Padding** in the case of the latter.

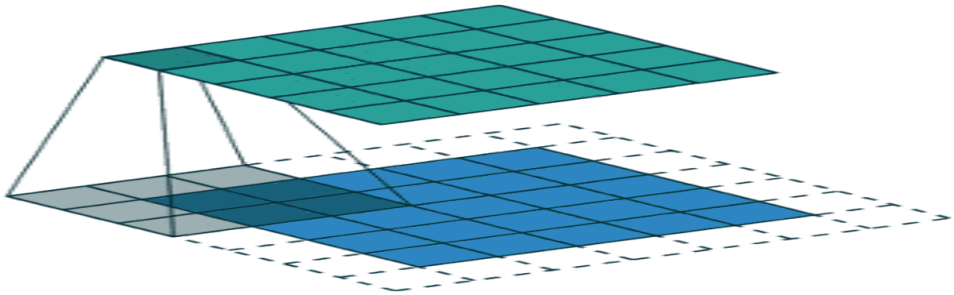


Figure 15: Applying Padding in the case of the latter.

SAME padding: 5x5x1 image is padded with 0s to create a 6x6x1 image

When we augment the 5x5x1 image into a 6x6x1 image and then apply the 3x3x1 kernel over it, we find that the convolved matrix turns out to be of dimensions 5x5x1. Hence the name — **Same Padding**.

On the other hand, if we perform the same operation without padding, we are presented with a matrix which has dimensions of the Kernel (3x3x1) itself — **Valid Padding**.

The following repository houses many such GIFs which would help you get a better understanding of how Padding and Stride Length work together to achieve results relevant to our needs.

Pooling Layer

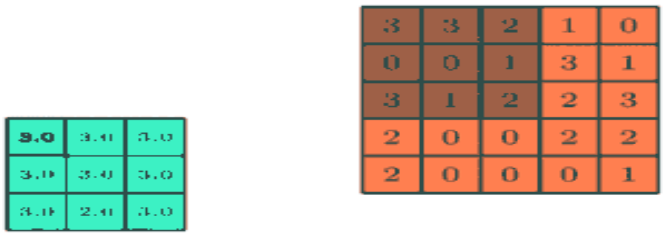


Figure 16: 3x3 pooling over 5x5 convolved feature

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to **decrease the computational power required to process the data** through dimensionality reduction. Furthermore, it is useful for **extracting dominant features** which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

There are two types of Pooling: Max Pooling and Average Pooling. **Max Pooling** returns the **maximum value** from the portion of the image covered by the Kernel. On the other hand, **Average Pooling** returns the **average of all the values** from the portion of the image covered by the Kernel.

Max Pooling also performs as a **Noise Suppressant**. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that **Max Pooling performs a lot better than Average Pooling**.

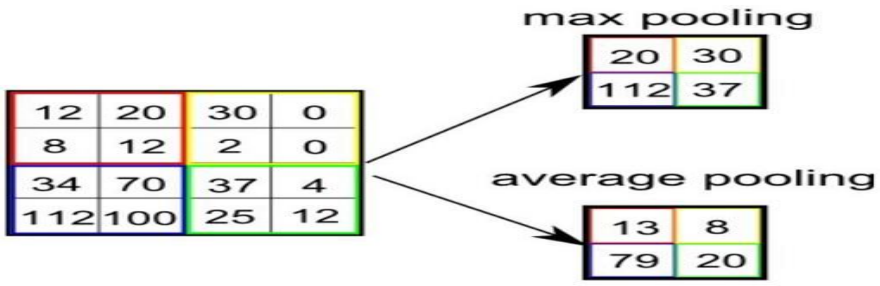


Figure 17: Image showing that max pooling is better than average pooling

Types of Pooling

The Convolutional Layer and the Pooling Layer, together form the i-th layer of a Convolutional Neural Network. Depending on the complexities in the images, the

number of such layers may be increased for capturing low-levels details even further, but at the cost of more computational power.

After going through the above process, we have successfully enabled the model to understand the features. Moving on, we are going to flatten the final output and feed it to a regular Neural Network for classification purposes.

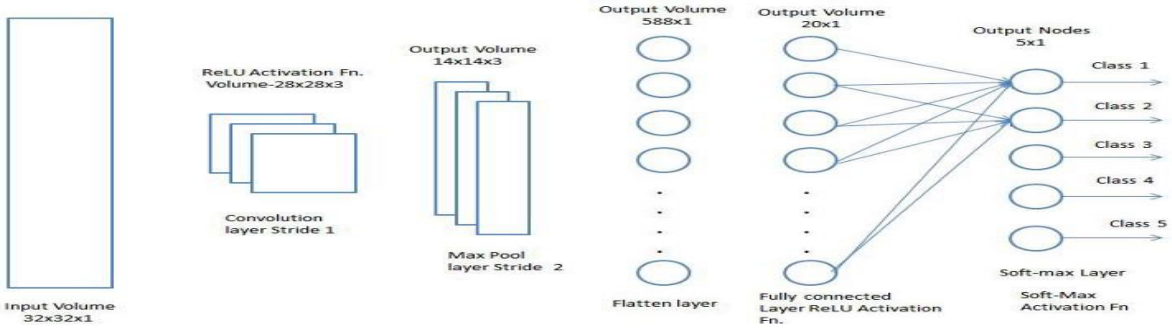


Figure 18: Classification — Fully Connected Layer (FC Layer)

Classification — Fully Connected Layer (FC Layer)

Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.

Now that we have converted our input image into a suitable form for our Multi-Level Perceptron, we shall flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the **Softmax Classification** technique.

2.RNN- Recurrent neural networks, or RNNs, are a type of artificial neural network that add additional weights to the network to create cycles in the network graph in an effort to maintain an internal state.

The promise of adding state to neural networks is that they will be able to explicitly learn and exploit context in sequence prediction problems, such as problems with an order or temporal component.

Overview

We will start off by setting the scene for the field of recurrent neural networks.

Next, we will take a closer look at LSTMs, GRUs, and NTM used for deep learning.

We will then spend some time on advanced topics related to using RNNs for deep learning.

- Recurrent Neural Networks
- Fully Recurrent Networks
- Recursive Neural Networks
- Long Short-Term Memory Networks
- Recurrent Neural Networks

Fully Recurrent Networks

The layered topology of a multilayer Perceptron is preserved, but every element has a weighted connection to every other element in the architecture and has a single feedback connection to itself.

Not all connections are trained and the extreme non-linearity of the error derivatives means conventional Back-propagation will not work, and so Back-propagation Through Time approaches or Stochastic Gradient Descent is employed.

Recursive Neural Networks

Recurrent neural networks are linear architectural variant of recursive networks.

Recursion promotes branching in hierarchical feature spaces and the resulting network architecture mimics this as training proceeds.

Training is achieved with Gradient Descent by sub-gradient methods.

Long Short-Term Memory Networks

With conventional Back-Propagation Through Time (BPTT) or Real Time Recurrent Learning (RTTL), error signals flowing backward in time tend to either explode or vanish.

The temporal evolution of the back-propagated error exponentially depends on the size of the weights. Weight explosion may lead to oscillating weights, while in vanishing causes learning to bridge long time lags and takes a prohibitive amount of time, or does not work at all.

LSTM is a novel recurrent network architecture training with an appropriate gradient-based learning algorithm.

LSTM is designed to overcome error back-flow problems. It can learn to bridge time intervals in excess of 1000 steps.

This true in presence of noisy, incompressible input sequences, without loss of short time lag capabilities.

Error back-flow problems are overcome by an efficient, gradient-based algorithm for an architecture enforcing constant (thus neither exploding nor vanishing) error flow through internal states of special units. These units reduce the effects of the "Input Weight Conflict" and the "Output Weight Conflict."

The Input Weight Conflict: Provided the input is non-zero, the same incoming weight has to be used for both storing certain inputs and ignoring others, then will often receive conflicting weight update signals.

These signals will attempt to make the weight participate in storing the input and protecting the input. This conflict makes learning difficult and calls for a more context-sensitive mechanism for controlling "write operations" through input weights.

The Output Weight Conflict: As long as the output of a unit is non-zero, the weight on the output connection from this unit will attract conflicting weight update signals generated during sequence processing.

These signals will attempt to make the outgoing weight participate in accessing the information stored in the processing unit and, at different times, protect the subsequent unit from being perturbed by the output of the unit being fed forward.

These conflicts are not specific to long-term lags and can equally impinge on short-term lags. Of note though is that as lag increases, stored information must be protected from perturbation, especially in the advanced stages of learning.

Network Architecture: Different types of units may convey useful information about the current state of the network. For instance, an input gate (output gate) may use inputs from other memory cells to decide whether to store (access) certain information in its memory cell.

Memory cells contain gates. Gates are specific to the connection they mediate. Input gates work to remedy the Input Weight Conflict while Output Gates work to eliminate the Output Weight Conflict.

Gates: Specifically, to alleviate the input and output weight conflicts and perturbations, a multiplicative input gate unit is introduced to protect the memory contents stored from perturbation by irrelevant inputs and a multiplicative output gate unit protects other units from perturbation by currently irrelevant memory contents stored.

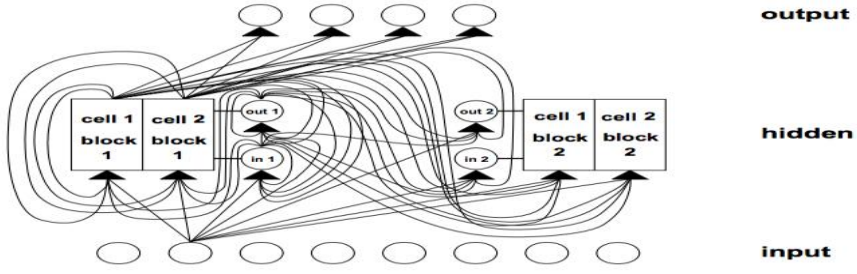


Figure 19: Example of LSTM Net

Example of an LSTM net with 8 input units, 4 output units, and 2 memory cell blocks of size 2. in1 marks the input gate, out1 marks the output gate, and cell1 = block1 marks the first memory cell of block 1.

Connectivity in LSTM is complicated compared to the multilayer Perceptron because of the diversity of processing elements and the inclusion of feedback connections.

Memory cell blocks: Memory cells sharing the same input gate and the same output gate form a structure called a "memory cell block".

Memory cell blocks facilitate information storage; as with conventional neural nets, it is not so easy to code a distributed input within a single cell. A memory cell block of size 1 is just a simple memory cell.

Learning: A variant of Real Time Recurrent Learning (RTRL) that takes into account the altered, multiplicative dynamics caused by input and output gates is used to ensure non-decaying error back propagated through internal states of memory cells errors arriving at "memory cell net inputs" do not get propagated back further in time.

Guessing: This stochastic approach can outperform many term lag algorithms. It has been established that many long-time lag tasks used in previous work can be solved more quickly by simple random weight guessing than by the proposed algorithms.

LSTM Limitations

- The efficient, truncated version of LSTM will not easily solve problems similar to "strongly delayed XOR."
- Each memory cell block needs an input gate and an output gate. Not necessary in other recurrent approaches.
- Constant error flow through "Constant Error Carrousels" inside memory cells produces the same effect as a conventional feed-forward architecture being presented with the entire input string at once.
- LSTM is as flawed with the concept of "regency" as other feed-forward approaches. Additional counting mechanisms may be required if fine-precision counting time steps is needed.

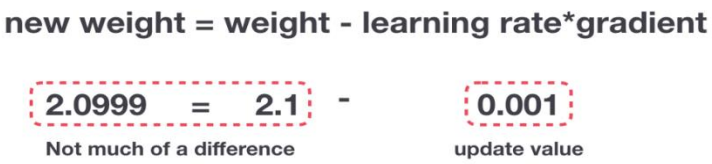
LSTM Advantages

- The algorithms ability to bridge long time lags is the result of constant error Backpropagation in the architecture's memory cells.
- LSTM can approximate noisy problem domains, distributed representations, and continuous values.
- LSTM generalizes well over problem domains considered. This is important given some tasks are intractable for already established recurrent networks.
- Fine tuning of network parameters over the problem domains appears to be unnecessary.

- In terms of update complexity per weight and time steps, LSTM is essentially equivalent to BPTT.
- LSTMs are showing to be powerful, achieving state-of-the-art results in domains like machine translation.

Recurrent Neural Networks suffer from short-term memory. If a sequence is long enough, they'll have a hard time carrying information from earlier time steps to later ones. So if you are trying to process a paragraph of text to do predictions, RNN's may leave out important information from the beginning.

During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks weights. The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn't contribute too much learning.



Gradient Update Rule

So in recurrent neural networks, layers that get a small gradient update stops learning. Those are usually the earlier layers. So because these layers don't learn, RNN's can forget what it seen in longer sequences, thus having a short-term memory. If you want to know more about the mechanics of recurrent neural networks in general, you can read my previous post here.

LSTM's and GRU's as a solution

LSTM 's and GRU's were created as the solution to short-term memory. They have internal mechanisms called gates that can regulate the flow of information.

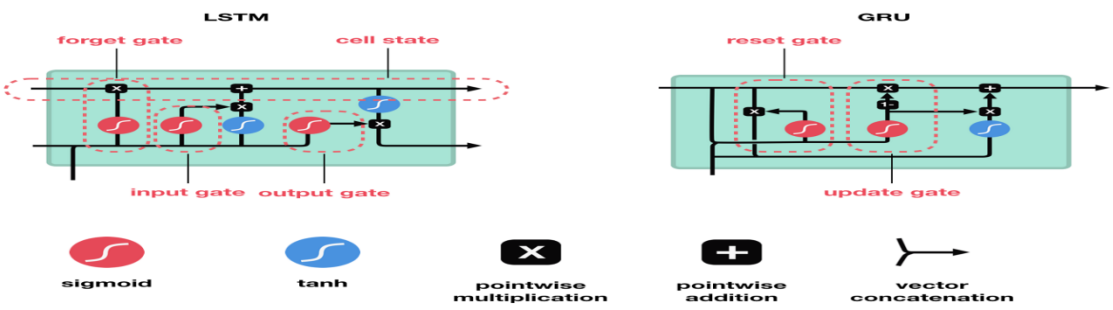


Figure 20: Image of LSTM and GRU

These gates can learn which data in a sequence is important to keep or throw away. By doing that, it can pass relevant information down the long chain of sequences to make predictions. Almost all state of the art results based on recurrent neural networks are achieved with these two networks. LSTM's and GRU's can be found in speech recognition, speech synthesis, and text generation.

Review of Recurrent Neural Networks

To understand how LSTM's or GRU's achieves this, let's review the recurrent neural network. An RNN works like this; First words get transformed into machine-readable vectors. Then the RNN processes the sequence of vectors one by one.

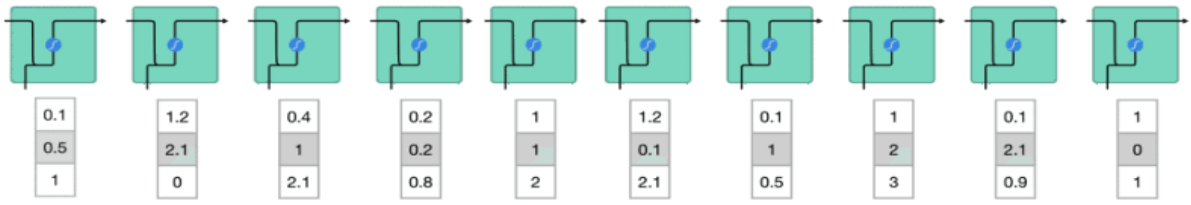


Figure 21: Processing sequence one by one

While processing, it passes the previous hidden state to the next step of the sequence. The hidden state acts as the neural networks memory. It holds information on previous data the network has seen before.

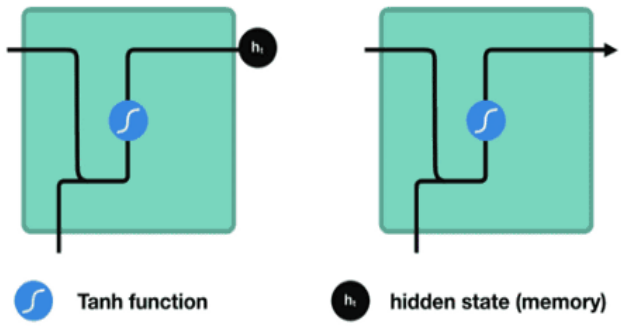


Figure 22: Image of Tanh function and Hidden state (memory)

Passing hidden state to next time step

Let's look at a cell of the RNN to see how you would calculate the hidden state. First, the input and previous hidden state are combined to form a vector. That vector now has information on the current input and previous inputs. The vector goes through the tanh activation, and the output is the new hidden state, or the memory of the network.

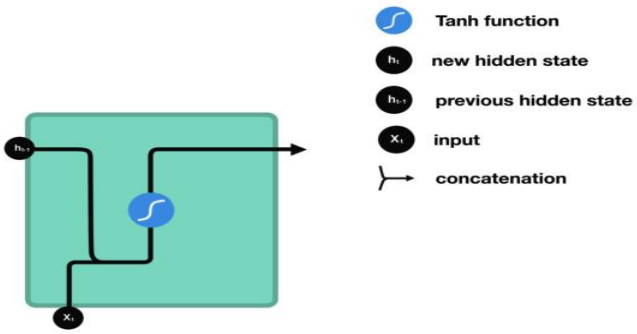


Figure 23: Passing hidden state to next time step

RNN Cell

Tanh activation

The tanh activation is used to help regulate the values flowing through the network. The tanh function squishes values to always be between -1 and 1.

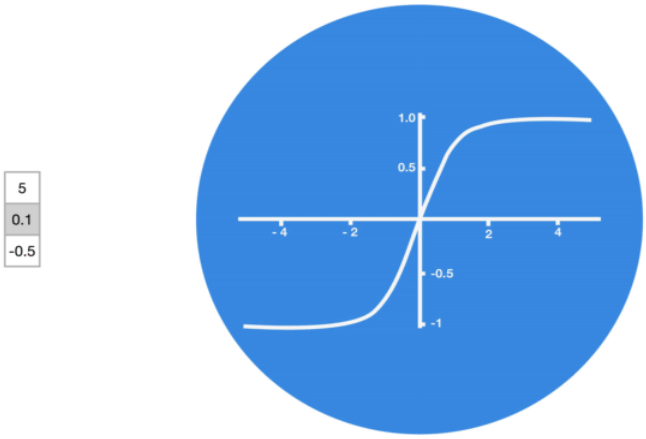


Figure 24: Tanh Activation function

Tanh squishes values to be between -1 and 1

When vectors are flowing through a neural network, it undergoes many transformations due to various math operations. So imagine a value that continues to be multiplied by let's say ***3***. You can see how some values can explode and become astronomical, causing other values to seem insignificant.

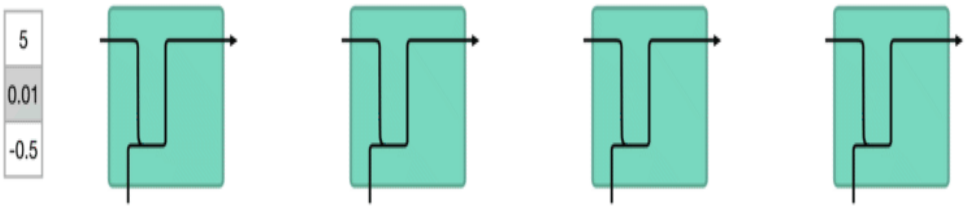


Figure 25: Vector transformations without tanh

A tanh function ensures that the values stay between -1 and 1, thus regulating the output of the neural network. You can see how the same values from above remain between the boundaries allowed by the tanh function.

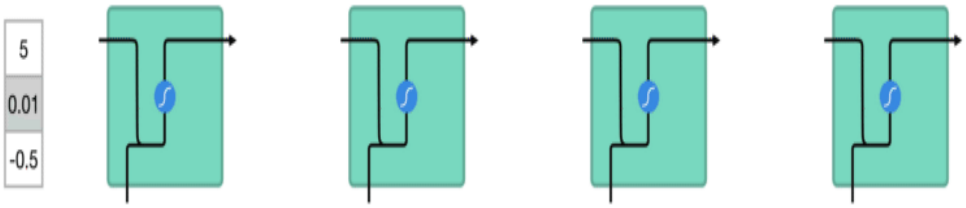


Figure 26: Vector transformations with tanh

So that's an RNN. It has very few operations internally but works pretty well given the right circumstances (like short sequences). RNN's uses a lot less computational resources than it's evolved variants, LSTM's and GRU's.

LSTM

An LSTM has a similar control flow as a recurrent neural network. It processes data passing on information as it propagates forward. The differences are the operations within the LSTM's cells.

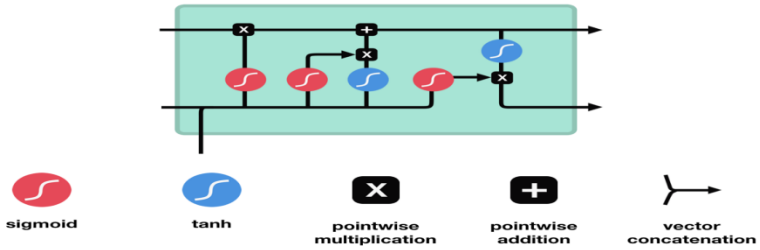


Figure 27: Image of an LSTM Cell

LSTM Cell and It's Operations

These operations are used to allow the LSTM to keep or forget information. Now looking at these operations can get a little overwhelming so we'll go over this step by step.

Core Concept

The core concept of LSTM's are the cell state, and it's various gates. The cell state act as a transport highway that transfers relative information all the way down the sequence chain. You can think of it as the "memory" of the network. The cell state, in theory, can carry relevant information throughout the processing of the sequence. So even information from the earlier time steps can make it's way to later time steps,

reducing the effects of short-term memory. As the cell state goes on its journey, information get's added or removed to the cell state via gates. The gates are different neural networks that decide which information is allowed on the cell state. The gates can learn what information is relevant to keep or forget during training.

Sigmoid

Gates contains sigmoid activations. A sigmoid activation is similar to the tanh activation. Instead of squishing values between -1 and 1, it squishes values between 0 and 1. That is helpful to update or forget data because any number getting multiplied by 0 is 0, causing values to disappears or be "forgotten." Any number multiplied by 1 is the same value therefore that value stay's the same or is "kept." The network can learn which data is not important therefore can be forgotten or which data is important to keep.

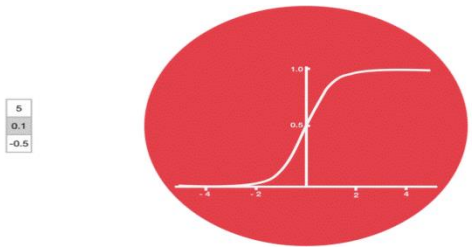


Figure 28: Sigmoid Function

Sigmoid squishes values to be between 0 and 1

Let's dig a little deeper into what the various gates are doing, shall we? So we have three different gates that regulate information flow in an LSTM cell. A forget gate, input gate, and output gate.

Forget gate

First, we have the forget gate. This gate decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.

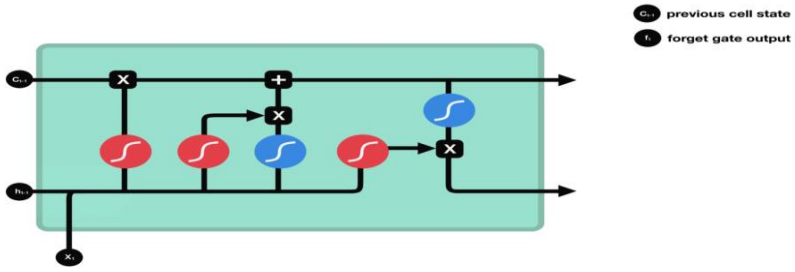


Figure 29: Forget Gate operation in LSTM cell

Forget gate operations

Input Gate

To update the cell state, we have the input gate. First, we pass the previous hidden state and current input into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important. You also pass the hidden state and current input into the tanh function to squish values between -1 and 1 to help regulate the network. Then you multiply the tanh output with the sigmoid output. The sigmoid output will decide which information is important to keep from the tanh output.

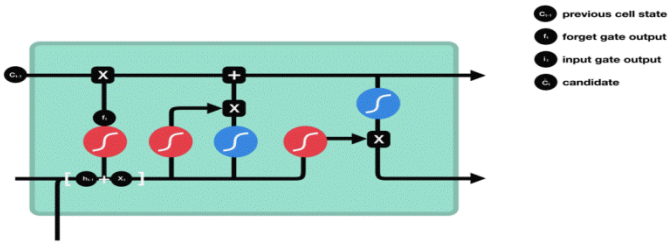


Figure 30:Input gate operations in LSTM Cell

Cell State

Now we should have enough information to calculate the cell state. First, the cell state gets pointwise multiplied by the forget vector. This has a possibility of dropping values in the cell state if it gets multiplied by values near 0. Then we take the output from the input gate and do a pointwise addition which updates the cell state to new values that the neural network finds relevant. That gives us our new cell state.

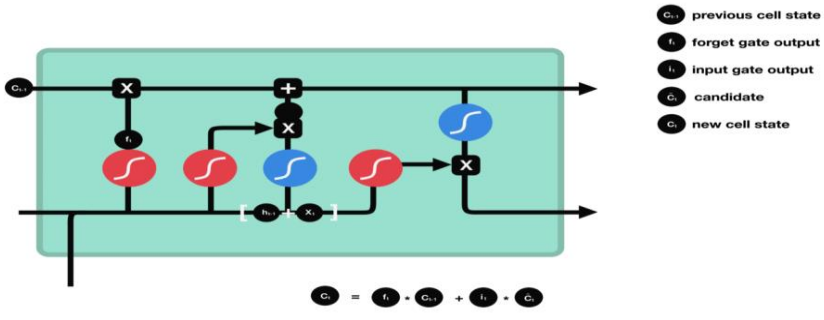


Figure 31: Calculation of cell state

Calculating cell state

Output Gate

Last we have the output gate. The output gate decides what the next hidden state should be. Remember that the hidden state contains information on previous inputs. The hidden state is also used for predictions. First, we pass the previous hidden state and the

current input into a sigmoid function. Then we pass the newly modified cell state to the tanh function. We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

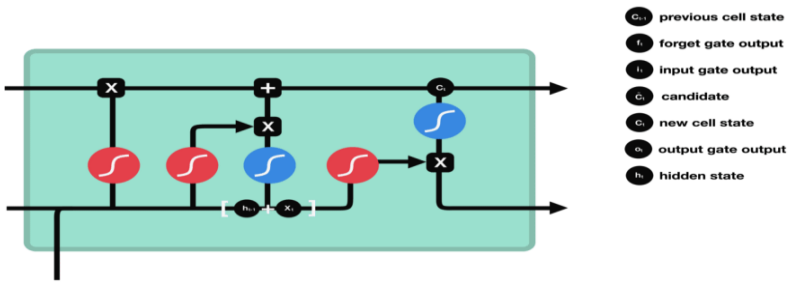


Figure 32:Output gate operations in lstm cell

output gate operations

To review, the Forget gate decides what is relevant to keep from prior steps. The input gate decides what information is relevant to add from the current step. The output gate determines what the next hidden state should be.

3.HAN- Since the uprising of Artificial Intelligence, text classification has become one of the most staggering tasks to accomplish. In layman terms, We can say Artificial Intelligence is the field which tries to achieve human-like intelligent models to ease the jobs for all of us. We have an astounding proficiency in text classification but even many sophisticated NLP models are failed to achieve proficiency even close to it. So the question arises is that what we humans do differently? How do we classify text?

First of all, we understand words not each and every word but many of them and we can guess even unknown words just by the structure of a sentence. Then we understand the message that those series of words (sentence) conveys. Then from those series of sentences, we understand the meaning of a paragraph or an article. The similar approach is used in Hierarchical Attention model.

So what's so special about this hierarchical thing?

Well to put it in a "too complicated to comprehend even for a techie" way, It uses stacked recurrent neural networks on word level followed by attention model to extract such words that are important to the meaning of the sentence and aggregate the representation of those informative words to form a sentence vector. Then the same procedure applied to the derived sentence vectors which then generate a vector who conceives the meaning of the given document and that vector can be passed further for text classification.

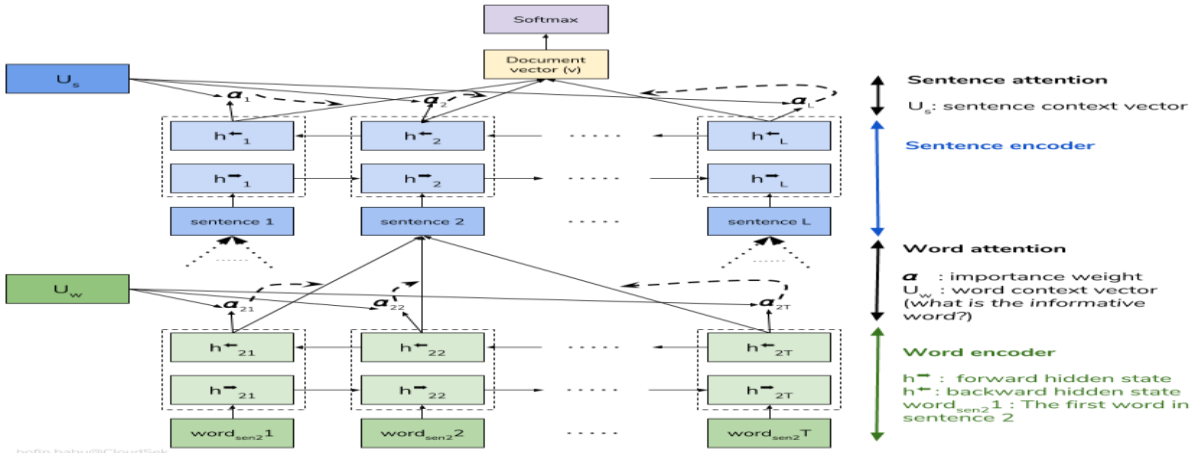


Figure 33: Structure of Hierarchical Attention Network Model

HAN Structure

The idea behind the paper is "Words make sentences and sentences make documents". The intent is to derive sentence meaning from the words and then derive the meaning of the document from those sentences. But not all words are equally important. Some of them characterize a sentence more than others. Therefore we use the attention model so that sentence vector can have more attention on "important" words. Attention model consists of two parts: Bidirectional RNN and Attention networks. While bidirectional RNN learns the meaning behind those sequence of words and returns vector corresponding to each word, Attention network gets weights corresponding to each word vector using its own shallow neural network. Then it aggregates the representation of those words to form a sentence vector i.e it calculates the weighted sum of every vector. This weighted sum embodies the whole sentence. The same procedure applies to sentence vectors so that the final vector embodies the gist of the whole document. Since it has two levels of attention model, therefore, it is called hierarchical attention networks.

Attention Model

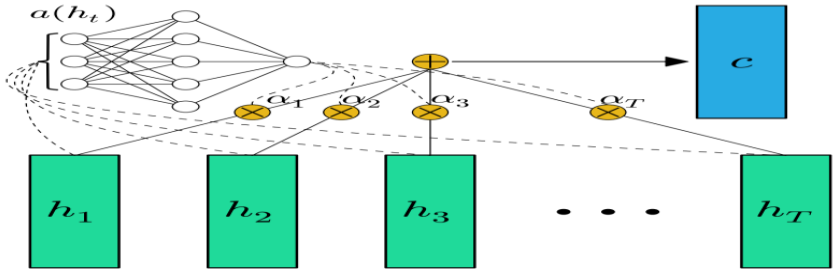


Figure 1: Schematic of our proposed "feed-forward" attention mechanism (cf. (Cho, 2015) Figure 1). Vectors in the hidden state sequence h_t are fed into the learnable function $a(h_t)$ to produce a probability vector α. The vector c is computed as a weighted average of h_t, with weighting given by α.

Figure 34: Schematic of our proposed "feed-forward" attention mechanism

The vectors from Bidirectional RNN pass through shallow neural network to decide weight corresponding to each vector. The weighted sum of each vector embodies the meaning of those vectors combined.

Data preprocessing

To process the data we need to convert it into a suitable form.

```
tokenizer = Tokenizer(num_words=max_features, oov_token=True)
tokenizer.fit_on_texts(texts)
data = np.zeros((len(texts), max_senten_num, max_senten_len),
dtype='int32')
for i, sentences in enumerate(paras):
    for j, sent in enumerate(sentences):
        if j< max_senten_num:
            wordTokens = text_to_word_sequence(sent)
            k=0
            for _, word in enumerate(wordTokens):
                try:
                    if k<max_senten_len and
tokenizer.word_index[word]<max_features:
                        data[i,j,k] = tokenizer.word_index[word]
                        k=k+1
                except:
                    print(word)
                    pass
```

We used the above code to convert training dataset to 3 dimension array: the first dimension represents the total number of documents, the second one represents each sentence in a document and the last one represents each word in a sentence. However, We have to set some upper limit in order to create a static graph which in this case are max_senten_len(max number of the sentence in a paragraph), max_senten_num(max number of words in a sentence) and max_features(max number of words Tokenizer can have).

Now isn't it unfair to the model if we randomly initialize all the words? Hence we use trained embedded vectors which give the model an extra edge in the terms of performance and yields better results.

```
GLOVE_DIR = "../input/glove6b/glove.6B.100d.txt"
embeddings_index = {}
f = open(GLOVE_DIR)
for line in f:
    try:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
    except:
        print(word)
        pass
f.close()
embedding_matrix = np.zeros((len(word_index) + 1, embed_size))
absent_words = 0
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
    else:
        absent_words += 1
print('Total absent words are', absent_words, 'which is', "%0.2f" %
(absent_words * 100 / len(word_index)), '% of total words')
```

We replaced the known words with their corresponding vector in embedding_matrix. Indeed, some words will be missing but our model has to learn to cope up with that.

Now time for HAN model

```
embedding_layer = Embedding(len(word_index) + 1,embed_size,weights=
[embedding_matrix], input_length=max_senten_len, trainable=False)

# Words level attention model
word_input = Input(shape=(max_senten_len,), dtype='float32')
word_sequences = embedding_layer(word_input)
word_lstm = Bidirectional(LSTM(150, return_sequences=True,
kernel_regularizer=l2_reg))(word_sequences)
word_dense = TimeDistributed(Dense(200, kernel_regularizer=l2_reg))
(word_lstm)
word_att = AttentionWithContext()(word_dense)
wordEncoder = Model(word_input, word_att)

# Sentence level attention model
sent_input = Input(shape=(max_senten_num, max_senten_len),
dtype='float32')
sent_encoder = TimeDistributed(wordEncoder)(sent_input)
sent_lstm = Bidirectional(LSTM(150, return_sequences=True,
kernel_regularizer=l2_reg))(sent_encoder)
sent_dense = TimeDistributed(Dense(200, kernel_regularizer=l2_reg))
(sent_lstm)
sent_att = Dropout(0.5)(AttentionWithContext()(sent_dense))
preds = Dense(30, activation='softmax')(sent_att)
model = Model(sent_input, preds)
model.compile(loss='categorical_crossentropy',optimizer='adam',metric
s=['acc'])
```

For word2vec, we used Keras Embedding layer. Time Distributed method is used to apply a Dense layer to each of the time-steps independently. We used Dropout and l2_reg regularizers to reduce overfitting.

4. RMDL-Random Multimodel Deep Learning is a new ensemble, deep learning approach for classification. Deep learning models have achieved state-of-the-art results across many domains. RMDL solves the problem of finding the best deep learning structure and architecture while simultaneously improving robustness and accuracy through ensembles of deep learning architectures. RDML can accept as input a variety data to include text, video, images, and symbolic.

RMDL includes 3 Random models, one DNN classifier at left, one Deep CNN classifier at middle, and one Deep RNN classifier at right (each unit could be LSTM or GRU).

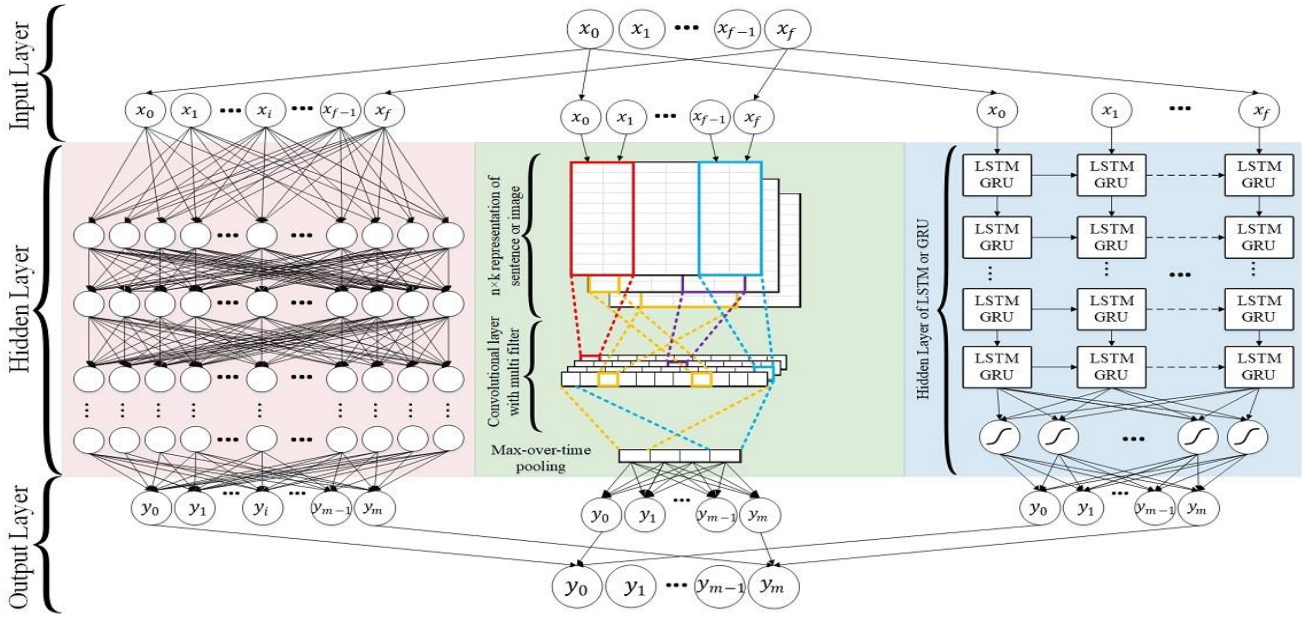


Figure 35:Structure of Random Multi-model for Deep Learning

The exponential growth in the number of complex datasets every year requires more enhancement in machine learning methods to provide robust and accurate data classification. Lately, deep learning approaches have been achieved surpassing results in comparison to previous machine learning algorithms on tasks such as image classification, natural language processing, face recognition, and etc. The success of these deep learning algorithms relys on their capacity to model complex and non-linear relationships within data. However, finding the suitable structure for these models has been a challenge for researchers. This paper introduces Random Multi-model Deep Learning (RMDL): a new ensemble, deep learning approach for classification. RMDL solves the problem of finding the best deep learning structure and architecture while simultaneously improving robustness and accuracy through ensembles of deep learning architectures. In short, RMDL trains multiple models of Deep Neural Network (DNN), Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) in parallel and combines their results to produce better result of any of those models individually. To create these models, each deep learning model has been constructed in a random fashion regarding the number of layers and nodes in their neural network structure. The resulting RDML model can be used for various domains such as text, video, images, and symbolic. In this Project, we describe RMDL model in depth and show the results for image and text classification as well as face recognition. For image classification, we compared our model with some of the available baselines using MNIST and CIFAR-10 datasets. Similarly, we used four datasets namely, WOS, Reuters, IMDB, and 20newsgroup and compared our results with available baselines. Web of Science (WOS) has been collected by authors and consists of three sets~(small, medium and large set). Lastly, we used ORL dataset to compare the performance of our approach with other face recognition methods. These test results show that RDML model consistently outperform standard methods over a broad range of data types and classification problems.

3.System Analysis and Design

3.1 Software and Hardware Requirements

Software Requirements

- Operating System : windows 7, windows vista, windows xp,windows 8, Windows 10

- Language : Python and Deep Learning

Hardware requirements

- Ram : 1 GB Ram and more
- Processor : Any Intel Processor
- HardDisk : 6 GB and more
- Speed : 1GHZ and more

3.2 Using COLAB-

What is Colaboratory?

Colaboratory, or "Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs.

Seems too good to be true. What are the limitations?

Colab resources are not guaranteed and not unlimited, and the usage limits sometimes fluctuate. This is necessary for Colab to be able to provide resources for free.

Users who are interested in more reliable access to better resources may be interested in Colab Pro.

What is the difference between Jupyter and Colab?

Jupyter is the open source project on which Colab is based. Colab allows you to use and share Jupyter notebooks with others without having to download, install, or run anything.

Using Colab

Where are my notebooks stored, and can I share them?

Colab notebooks are stored in Google Drive, or can be loaded from GitHub. Colab notebooks can be shared just as you would with Google Docs or Sheets. Simply click the Share button at the top right of any Colab notebook

If I share my notebook, what will be shared?

If you choose to share a notebook, the full contents of your notebook (text, code, output, and comments) will be shared. You can omit code cell output from being saved or shared by using **Edit > Notebook settings > Omit code cell output when saving this notebook**. The virtual machine you're using, including any custom files and libraries that you've setup, will not be shared. So it's a good idea to include cells which install and load any custom libraries or files that your notebook needs.

Can I import an existing Jupyter/IPython notebook into Colab?

Yes. Choose "Upload notebook" from the File menu.

How can I search Colab notebooks?

You can search Colab notebooks using Google Drive. Clicking on the Colab logo at the top left of the notebook view will show all notebooks in Drive. You can also search for notebooks that you have opened recently using **File > Open Recent**.

Where is my code executed? What happens to my execution state if I close the browser window?

Code is executed in a virtual machine private to your account. Virtual machines are deleted when idle for a while, and have a maximum lifetime enforced by the Colab service.

How can I get my data out?

You can download any Colab notebook that you've created from Google Drive following these instructions, or from within Colab's File menu. All Colab notebooks are stored in the open source Jupyter notebook format (.ipynb).

How can I reset the virtual machine(s) my code runs on, and why is this sometimes unavailable?

Selecting **Runtime > Factory reset runtime** to return all managed virtual machines assigned to you to their original state. This can be helpful in cases where a virtual machine has become unhealthy e.g. due to accidental overwrite of system files, or installation of incompatible software. Colab limits how often this can be done to prevent undue resource consumption. If an attempt fails, please try again later.

Why aren't resources guaranteed in Colab?

In order to be able to offer computational resources for free, Colab needs to maintain the flexibility to adjust usage limits and hardware availability on the fly. Resources

available in Colab vary over time to accommodate fluctuations in demand, as well as to accommodate overall growth and other factors.

Some users want to be able to do more in Colab than the resource limits allow. We have heard from many users who want faster GPUs, longer running notebooks and more memory, as well as usage limits that are higher and don't fluctuate as much. Our long term goal is to continue providing a free version of Colab, while also growing in a sustainable fashion to meet the needs of our users.

What are the usage limits of Colab?

Colab is able to provide free resources in part by having dynamic usage limits that sometimes fluctuate, and by not providing guaranteed or unlimited resources. This means that overall usage limits as well as idle timeout periods, maximum VM lifetime, GPU types available, and other factors vary over time. Colab does not publish these limits, in part because they can (and sometimes do) vary quickly.

GPUs and TPUs are sometimes prioritized for users who use Colab interactively rather than for long-running computations, or for users who have recently used less resources in Colab. As a result, users who use Colab for long-running computations, or users who have recently used more resources in Colab, are more likely to run into usage limits and have their access to GPUs and TPUs temporarily restricted. Users with high computational needs may be interested in using Colab's UI with a local runtime running on their own hardware.

What types of GPUs are available in Colab?

The types of GPUs that are available in Colab vary over time. This is necessary for Colab to be able to provide access to these resources for free. The GPUs available in Colab often include Nvidia K80s, T4s, P4s and P100s. There is no way to choose what type of GPU you can connect to in Colab at any given time.

How long can notebooks run in Colab?

Notebooks run by connecting to virtual machines that have maximum lifetimes that can be as much as 12 hours. Notebooks will also disconnect from VMs when left idle for too long. Maximum VM lifetime and idle timeout behavior may vary over time, or based on your usage. This is necessary for Colab to be able to offer computational resources for free.

How much memory is available in Colab?

The amount of memory available in Colab virtual machines varies over time (but is stable for the lifetime of the VM). (Adjusting memory over time allows us to continue to offer Colab for free.) You may sometimes be automatically assigned a VM with extra memory when Colab detects that you are likely to need it.

3.3 Data Flow Diagrams

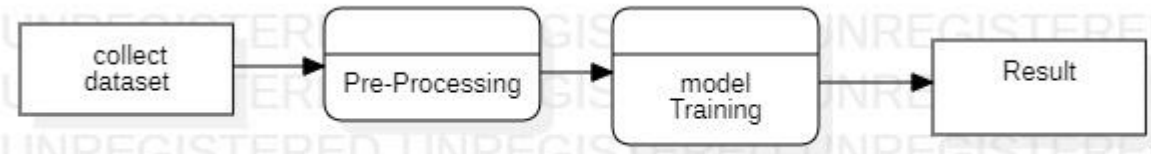


Figure 36: 0 Level Data Flow Diagram

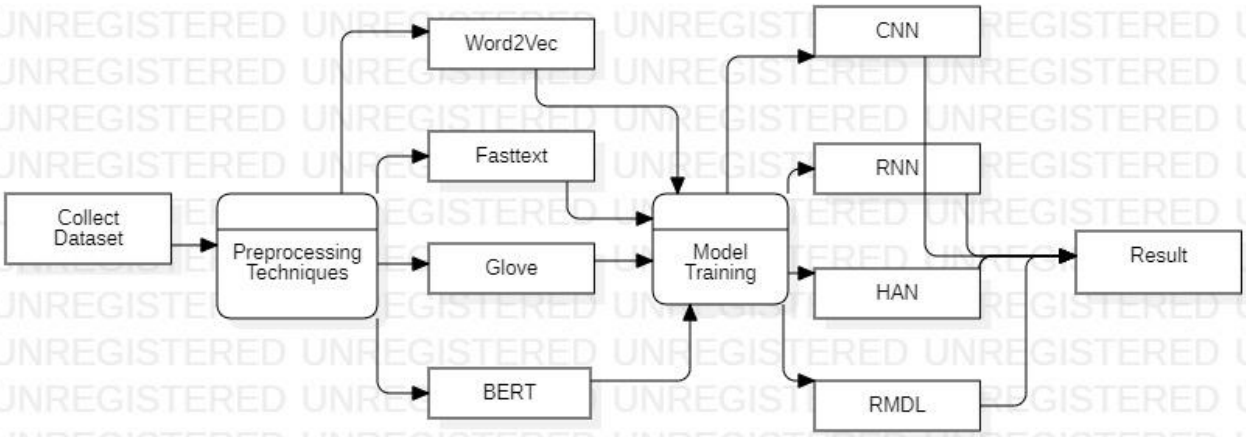


Figure 37: 1 Level Data Flow Diagram

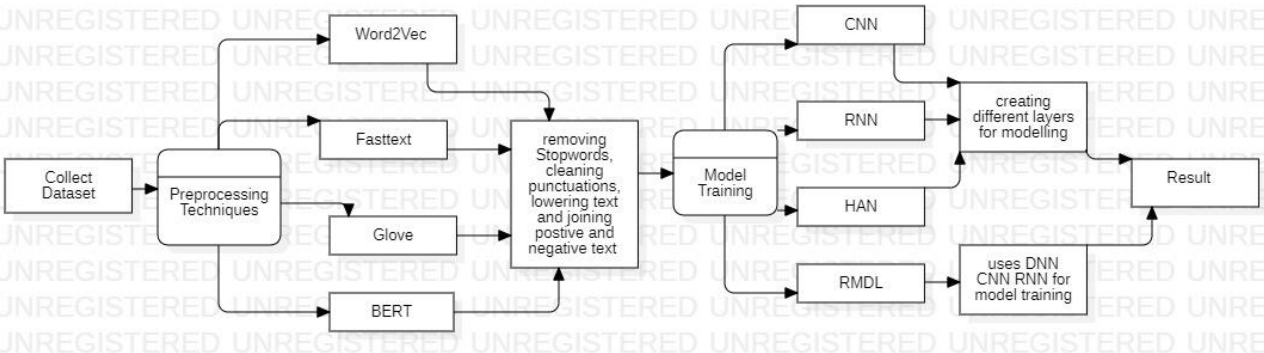


Figure 38: 2 Level Data Flow Diagram

4.Implementation

We used COLAB for the implementation of “Hotel Review Based Sentimental Analysis ”. There are several stages involved during implementation of our problem using different Preprocessing and Modeling techniques. Among them training and testing are the two main phases that are involved .

1. Cleaning and Converting the text into lower case

```
while ischar(line)
temp_line1=lower(line);
temp_line2=strrep(temp_line1,' ','');
temp_line3=strrep(temp_line2,',','');
temp_line4=strrep(temp_line3,';','');
temp_line5=strrep(temp_line4,':','');
temp_line6=strrep(temp_line5,'"','');
temp_line7=strrep(temp_line6,'\','');
temp_line8=strrep(temp_line7,')','');
temp_line9=strrep(temp_line8,'(','');
temp_line10=strrep(temp_line9,'?','');
temp_line11=strrep(temp_line10,'!','');
end
```

2.Shuffling the data and joining both positive and Negative Reviews

```
# Shuffling data
reviews_df = reviews_df.sample(frac=1).reset_index(drop=True)

# Extracting all text
positive_reviews = reviews_df['Positive_Review'].values
negative_reviews = reviews_df['Negative_Review'].values
reviews_text = []

for p,n in zip(positive_reviews, negative_reviews) :
    if p in ['na', 'nothing', 'none', 'n a', 'no', 'no positive', 'no negative'] :
        reviews_text.append(n)
    elif n in ['na', 'nothing', 'none', 'n a', 'no', 'no positive', 'no negative'] :
        reviews_text.append(p)
```

```
    else :
        reviews_text.append(n)
        reviews_text.append(p)
```

3.Creating Dictionary and dividing the dataset into reviews and labels

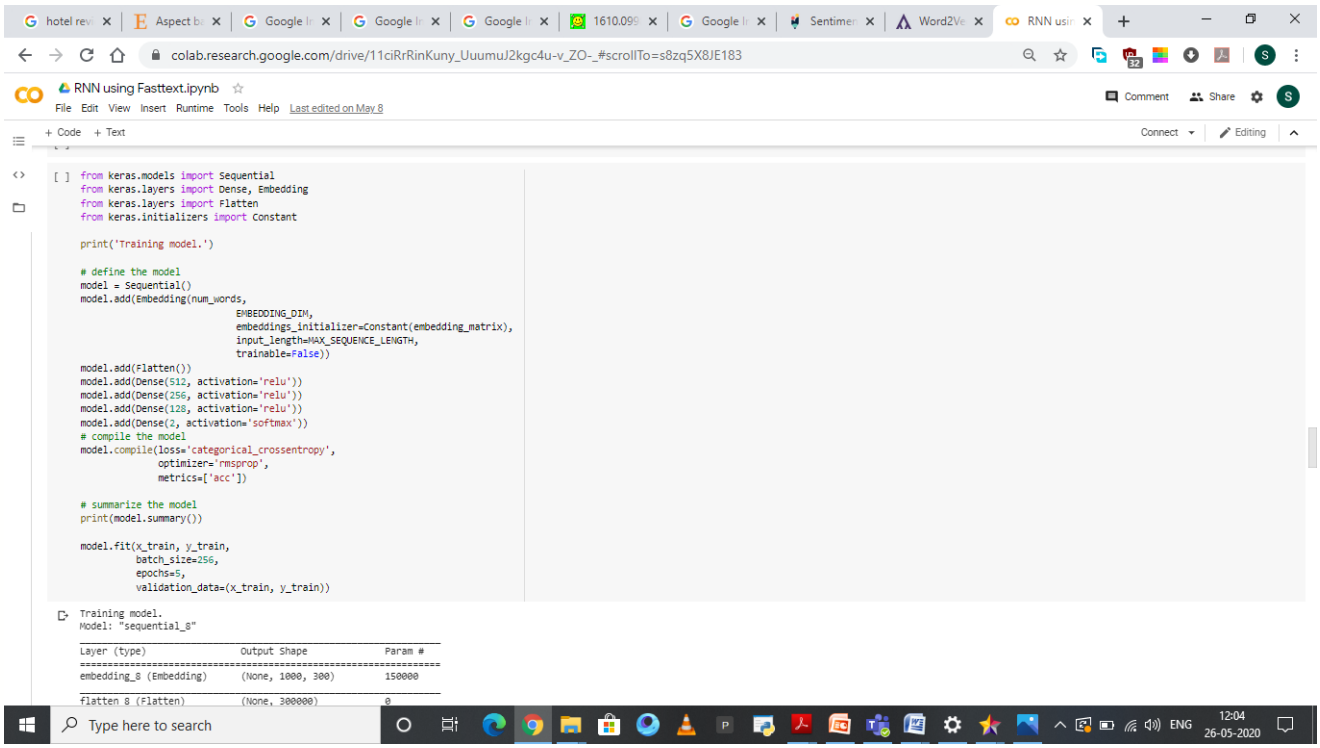
```
training_df = reviews_df.loc[:10000]
positive_reviews_filtered = training_df['Positive_Review'].values
negative_reviews_filtered = training_df['Negative_Review'].values
training_reviews = []
labels = []

for idx,(p,n) in enumerate(zip(positive_reviews_filtered, negative_reviews_filtered)) :
    if p in ['na', 'nothing', 'none', 'n a', 'no', 'no positive', 'no negative'] :
        training_reviews.append(n)
        labels.append(0)
    elif n in ['na', 'nothing', 'none', 'n a', 'no', 'no positive', 'no negative'] :
        training_reviews.append(p)
        labels.append(1)
    else :
        training_reviews.append(n)
        labels.append(0)
        training_reviews.append(p)
        labels.append(1)
```

4.Training and Testing

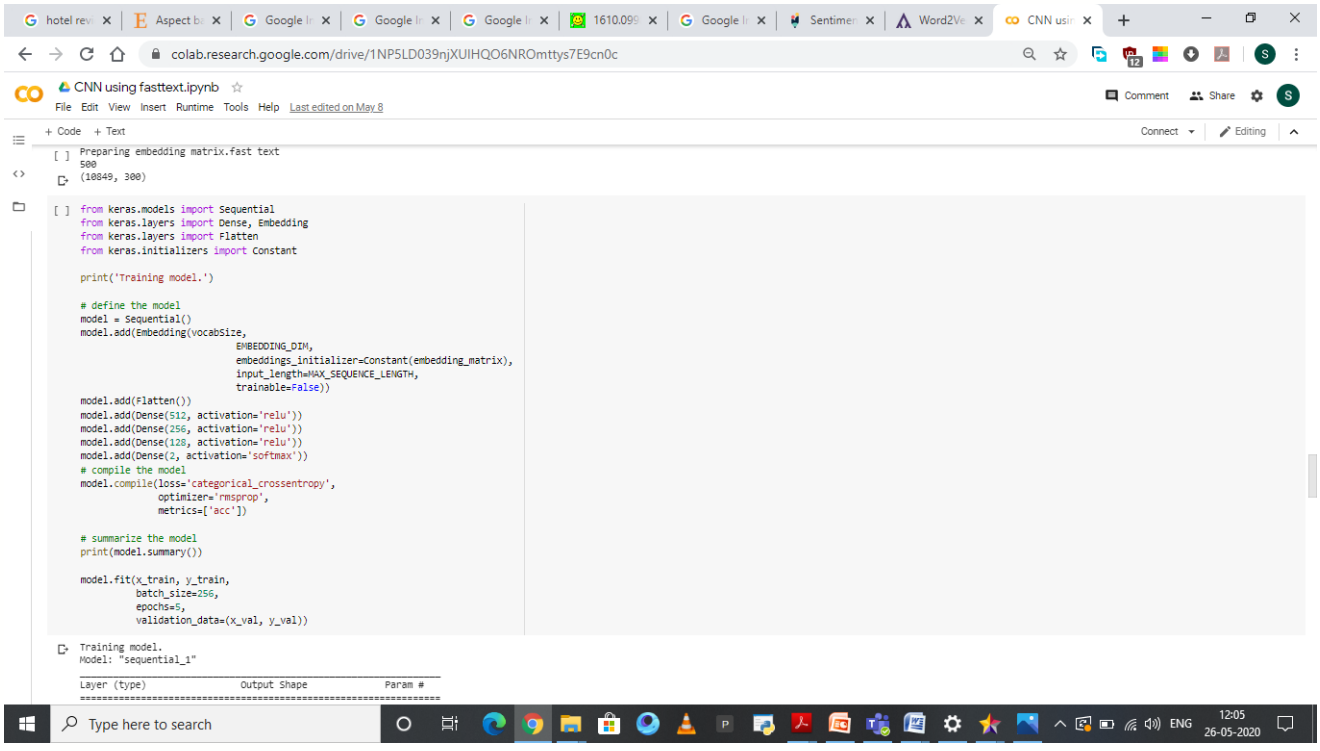
RNN using Fasttext
Pseudocode:

1. mounted the Dataset to Colab using Google Drive
2. read the dataset reviews_df ←text_reviews.csv
3. clean the text as def clean(text){};
4. Import tensorflow and keras
5. Create dictionary as dict={ text=”reviews”, sentiments=”labels”}
6. Create tokenizer
7. Split the data into training and test set(70:30) ratio
8. import wiki-news dataset for fasttext embedding
9. Create embedding matrix using fasttext
10. Then we train the RNN model having multiple layer
11. accuracy of trained model on testing dataset as accuracy ← model.evaluate()



CNN using Fasttext
Pseudocode:

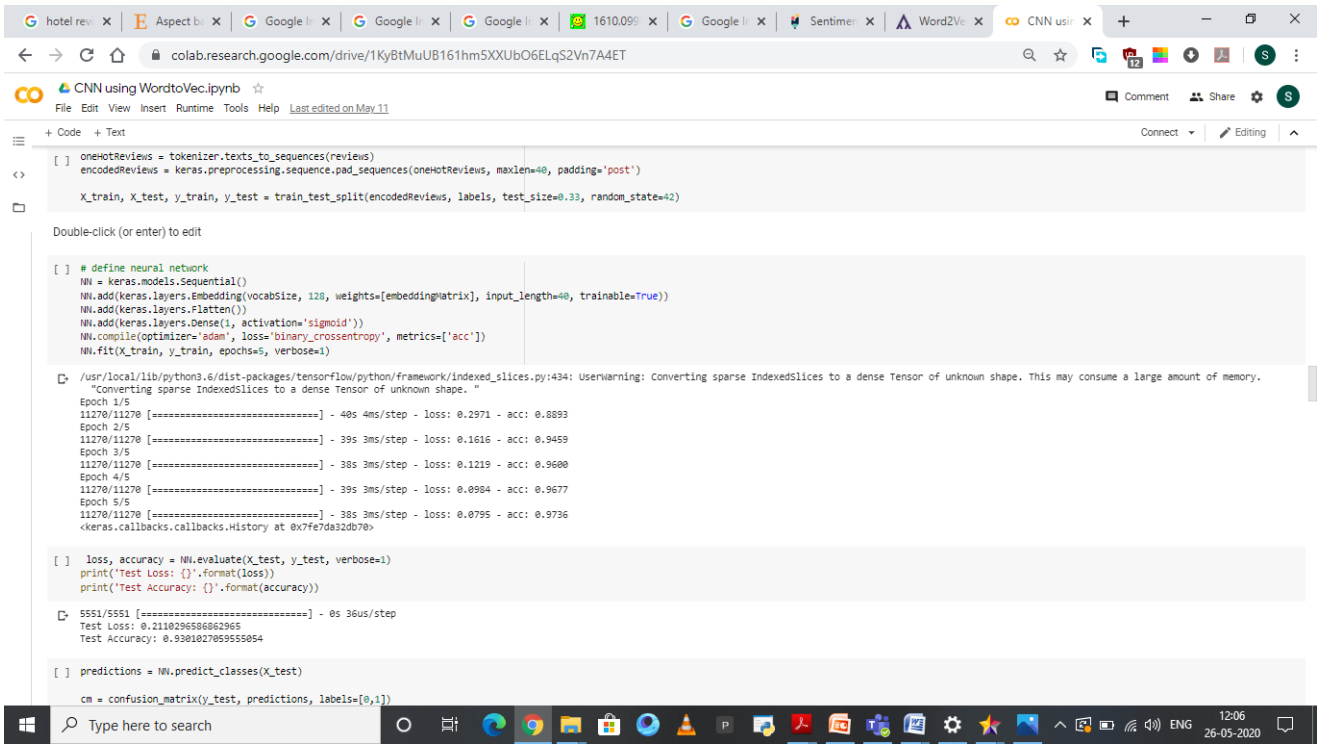
1. mounted the Dataset to Colab using Google Drive
2. read the dataset reviews_df ←text_reviews.csv
3. clean the text as def clean(text){};
4. Import tensorflow and keras
5. Create dictionary as dict={ text=”reviews”, sentiments=”labels”}
6. Create tokenizer
7. Split the data into training and test set(70:30) ratio
8. import wiki-news dataset for fasttext embedding
9. Create embedding matrix using fasttext
10. Then we train the CNN model having single layer
11. accuracy of trained model on testing dataset as accuracy ← model.evaluate()



CNN using Word2Vec

Pseudocode:

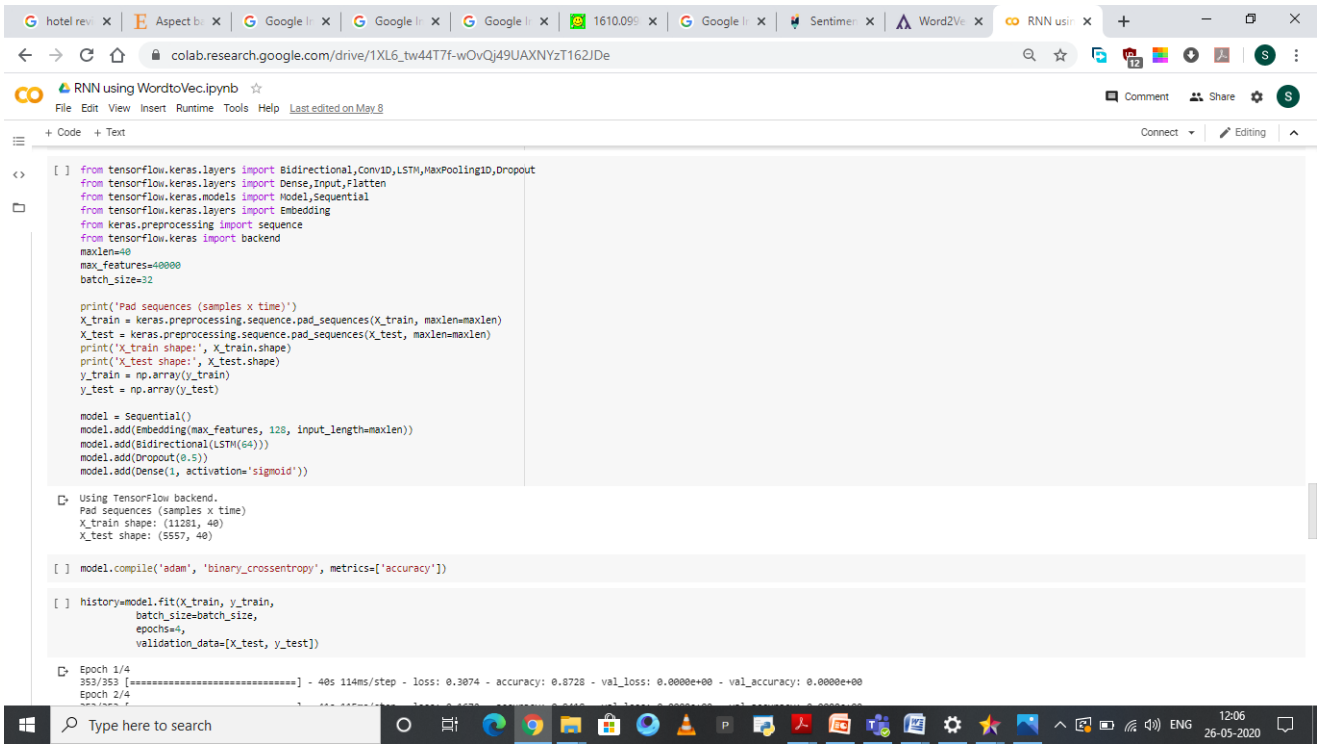
1. mounted the Dataset to Colab using Google Drive
2. read the dataset reviews_df ←text_reviews.csv
3. clean the text as def clean(text){};
4. Import tensorflow and keras
5. Create dictionary as dict={ text=”reviews”, sentiments=”labels”}
6. Create tokenizer
7. Split the data into training and test set(70:30) ratio
8. Create tokenizer using vocabsize for word2Vec embedding
9. Create embedding matrix using tokenizer
10. Then we train the CNN model having single layer
11. accuracy of trained model on testing dataset as accuracy ← model.evaluate()



RNN using Word2Vec

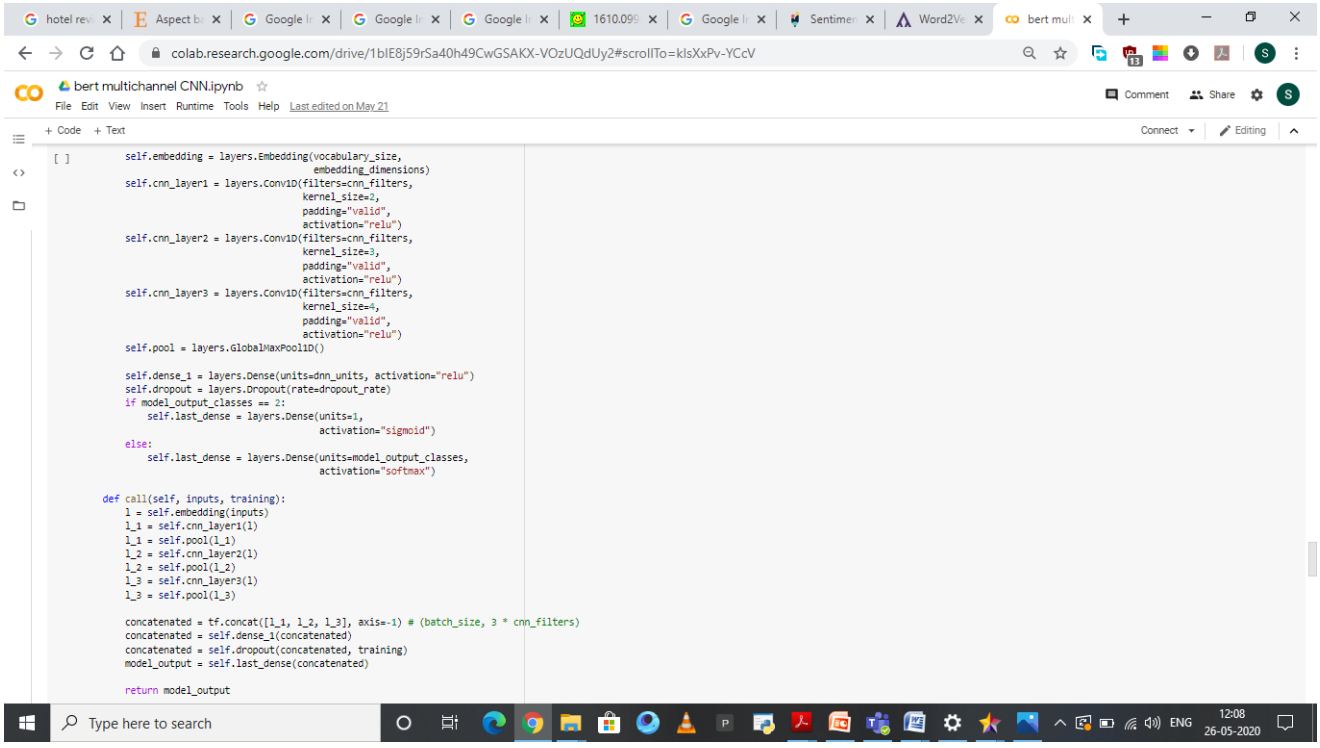
Pseudocode:

1. mounted the Dataset to Colab using Google Drive
2. read the dataset reviews_df ←text_reviews.csv
3. clean the text as def clean(text){};
4. Import tensorflow and keras
5. Create dictionary as dict={ text=”reviews”, sentiments=”labels”}
6. Create tokenizer
7. Split the data into training and test set(70:30) ratio
8. Create vocabsize using tokenizer
9. embedding matrix using Word2Vec tokenizer
10. Then we train the RNN model having multiple layer
11. accuracy of trained model on testing dataset as accuracy ← model.evaluate()



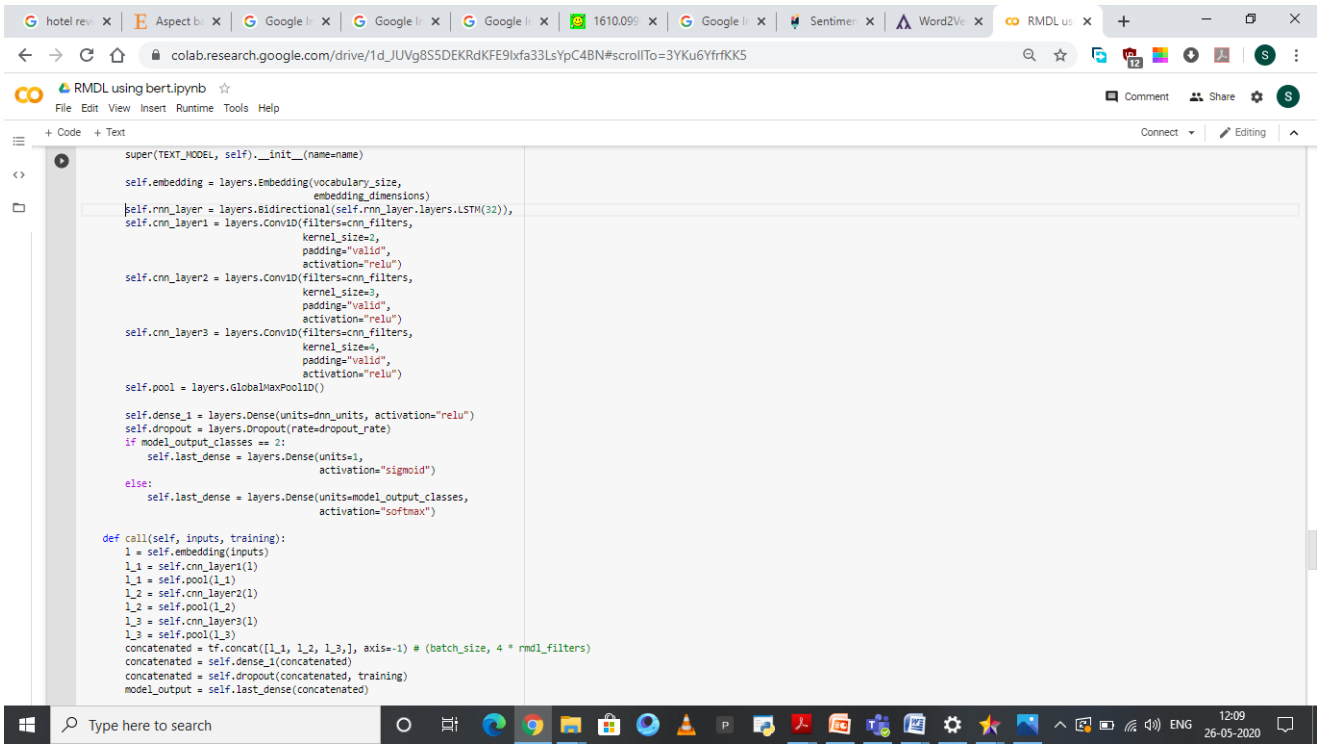
MCNN using BERT
Pseudocode:

1. mounted the Dataset to Colab using Google Drive
2. Install bert
3. read the dataset reviews_df ←text_reviews.csv
4. clean the text as def clean(text){};
5. Import tensorflow and keras
6. Create dictionary as dict={ text=”reviews”, sentiments=”labels”}
7. Create bert tokenizer
8. Shuffle the dataset
9. Create batched Dataset for model traing
10. Split the data into training and test set(70:30) ratio
11. train the MCNN model having multiple layer
12. accuracy of trained model on testing dataset as accuracy ← model.evaluate()



RMDL using BERT
Pseudocode:

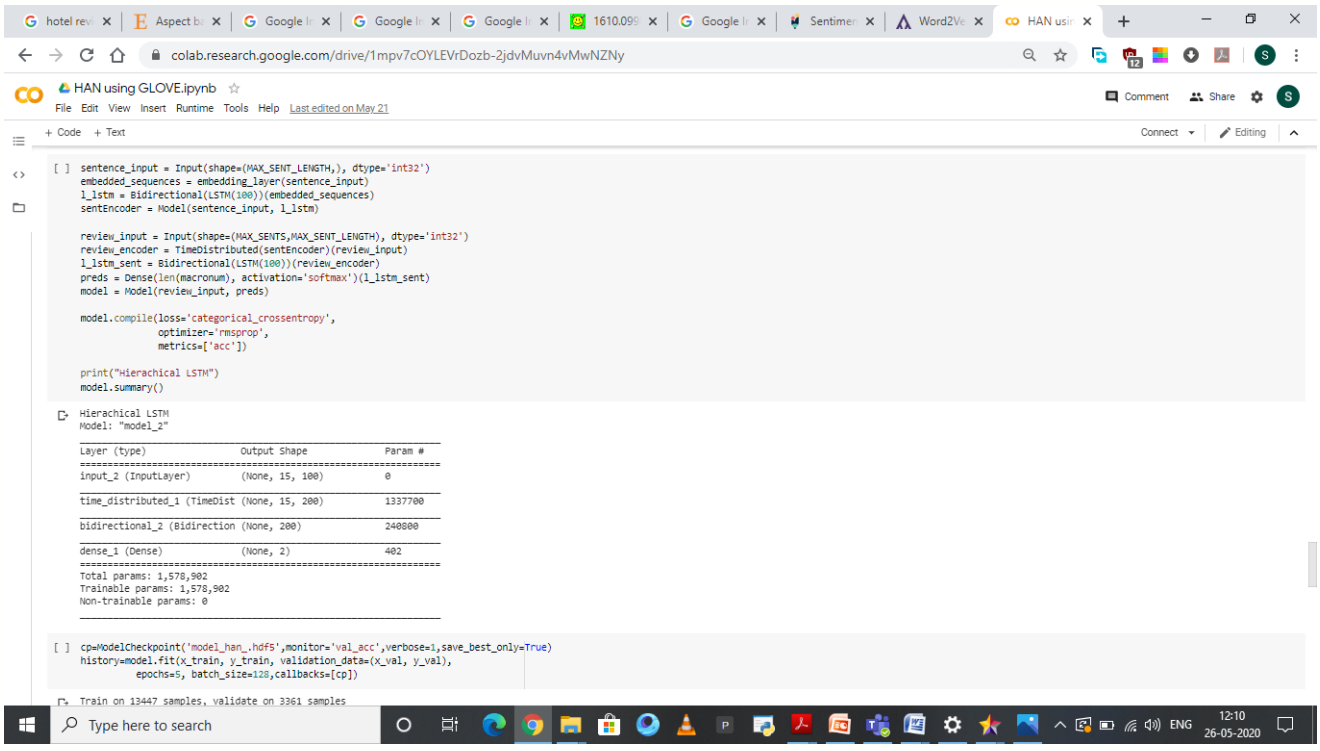
1. mounted the Dataset to Colab using Google Drive
2. Install bert
3. read the dataset reviews_df ←text_reviews.csv
4. clean the text as def clean(text){};
5. Import tensorflow and keras
6. Create dictionary as dict={ text=”reviews”, sentiments=”labels”}
7. Create bert tokenizer
8. Shuffle the dataset
9. Create batched Dataset for model traing
10. Split the data into training and test set(70:30) ratio
11. train the RMDL model having (DNN CNN RNN as three internal models)
12. accuracy of trained model on testing dataset as accuracy ← model.evaluate()



HAN using Glove

Pseudocode:

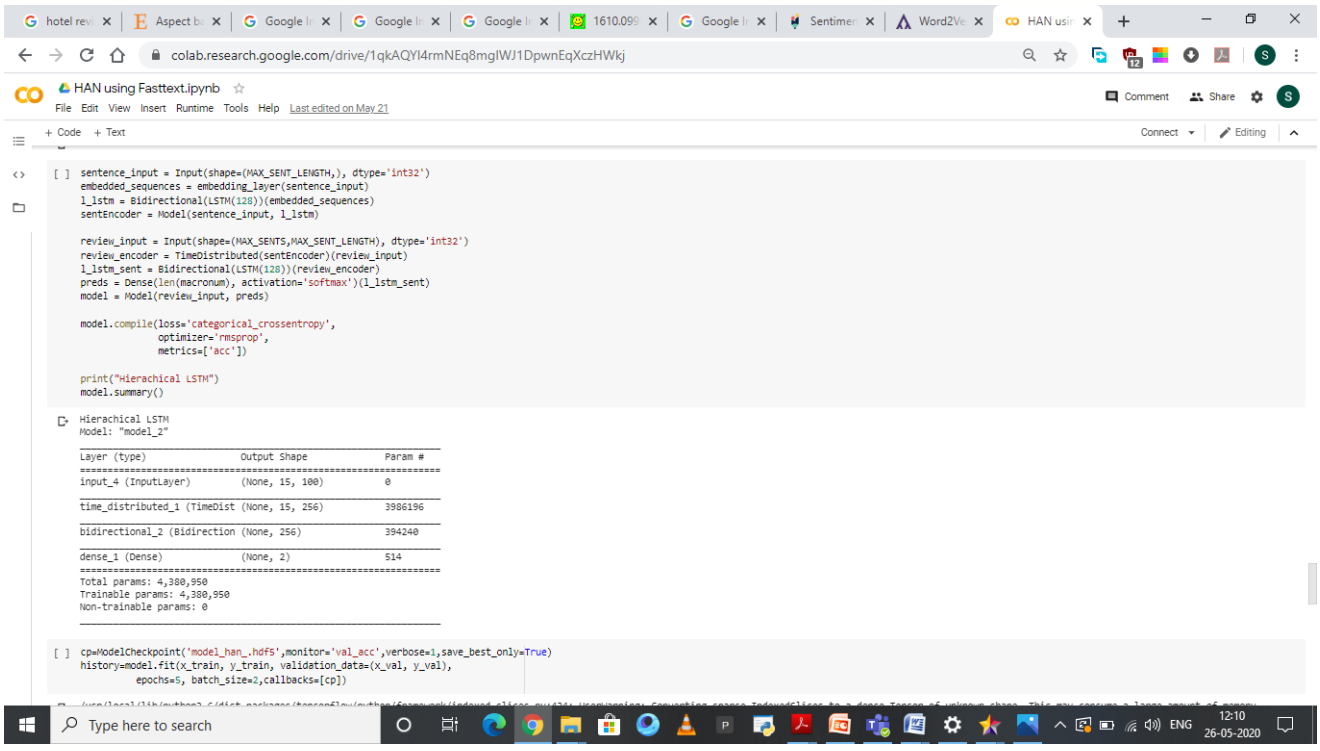
1. mounted the Dataset to Colab using Google Drive
2. read the dataset reviews_df ←text_reviews.csv
3. clean the text as def clean(text){};
4. Import theano and keras
5. Create dictionary as dict={ text=”reviews”, sentiments=”labels”}
6. Create tokenizer
7. Import glove.6B dataset for word vectors
8. Split the data into training and test set(70:30) ratio
9. Create sentence tokenizer and word tokenizer
10. Create embedding matrix using both tokenizer
11. Then we train the HAN model .
12. accuracy of trained model on testing dataset as accuracy ← model.evaluate()



HAN using Fasttext

Pseudocode:

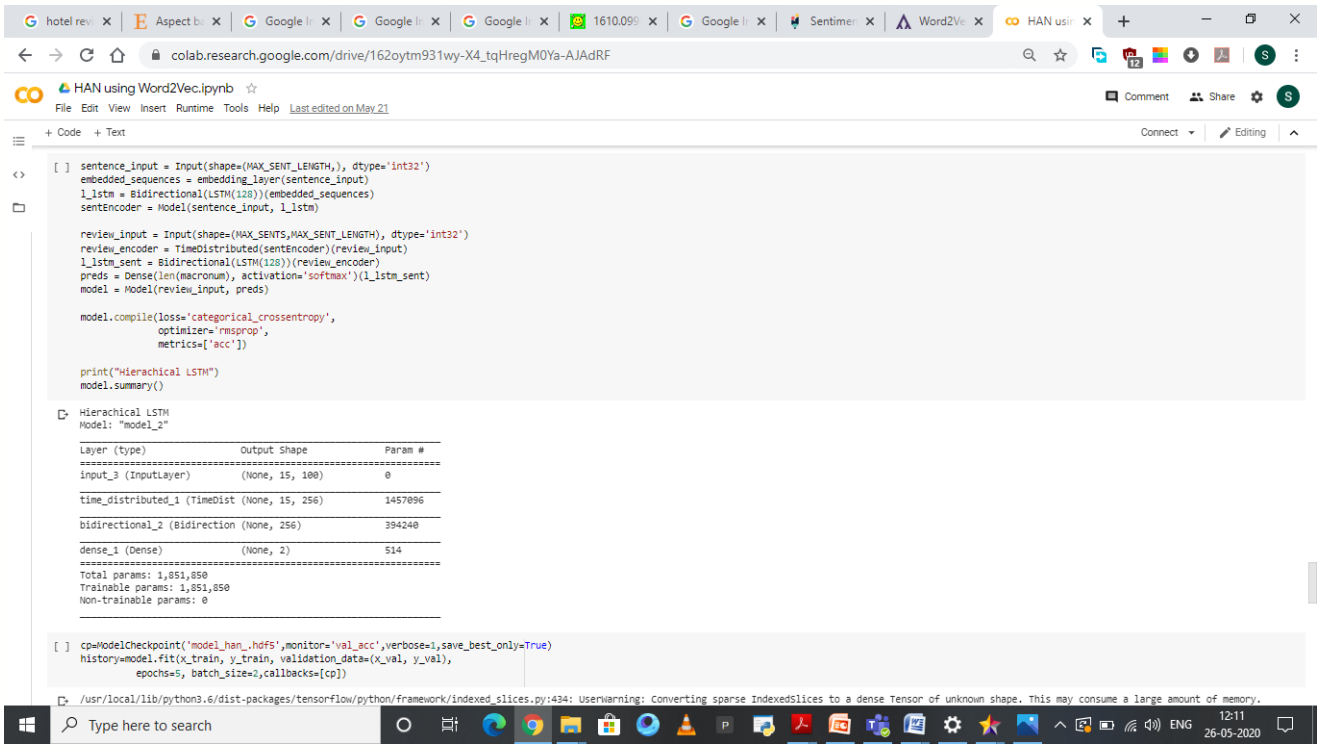
1. mounted the Dataset to Colab using Google Drive
2. read the dataset reviews_df ←text_reviews.csv
3. clean the text as def clean(text){};
4. Import theano and keras
5. Create dictionary as dict={ text=”reviews”, sentiments=”labels”}
6. Create tokenizer
7. Import wiki-news dataset for word embedding
8. Split the data into training and test set(70:30) ratio
9. Create sentence tokenizer and word tokenizer
10. Create embedding matrix using both tokenizer
11. Then we train the HAN model .
12. accuracy of trained model on testing dataset as accuracy ← model.evaluate()



HAN using Word2Vec

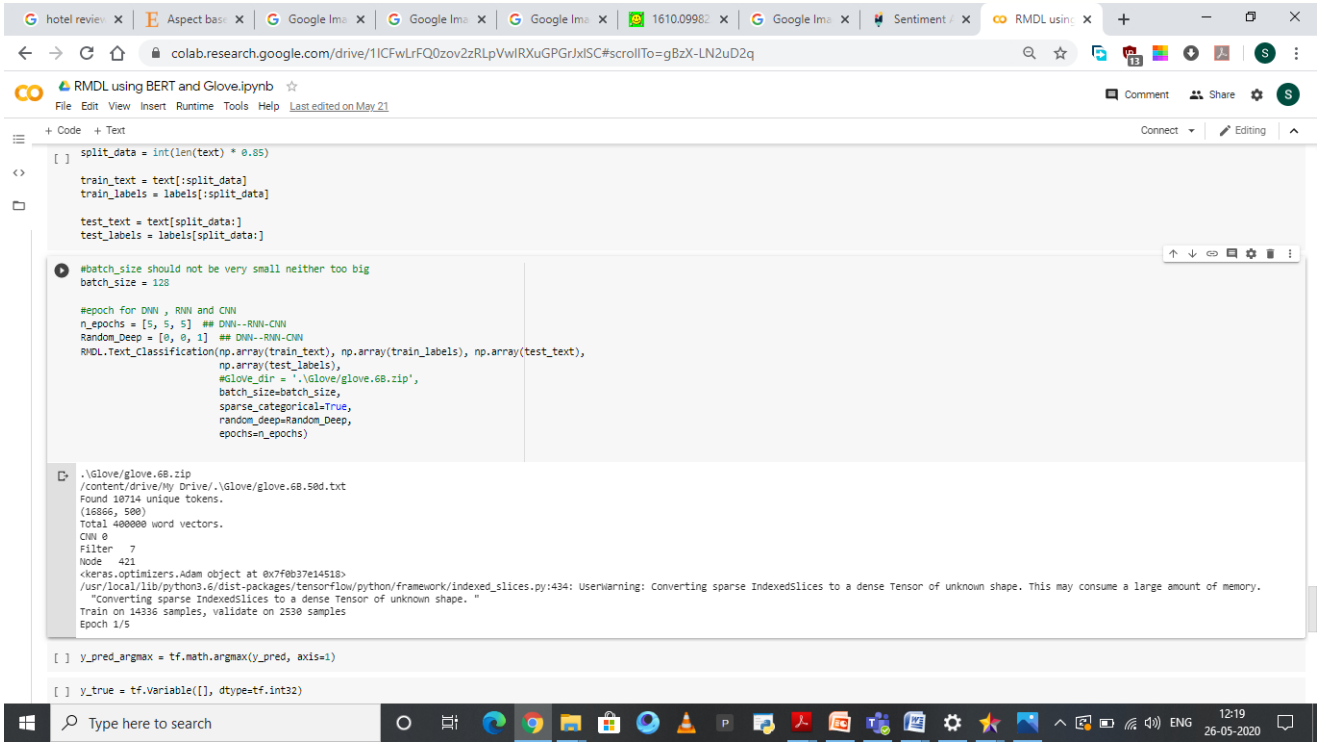
Pseudocode:

1. mounted the Dataset to Colab using Google Drive
2. read the dataset reviews_df ←text_reviews.csv
3. clean the text as def clean(text){};
4. Import theano and keras
5. Create dictionary as dict={ text=”reviews”, sentiments=”labels”}
6. Create tokenizer using Word2vec
7. Split the data into training and test set(70:30) ratio
8. Create sentence tokenizer and word tokenizer
9. Create embedding matrix using both tokenizer
10. Then we train the HAN model .
11. accuracy of trained model on testing dataset as accuracy ← model.evaluate()



RMDL using Glove
Pseudocode:

1. mounted the Dataset to Colab using Google Drive
2. Install RMDL
3. read the dataset reviews_df ←text_reviews.csv
4. clean the text as def clean(text){};
5. Import tensorflow and keras
6. Create dictionary as dict={ text=”reviews”, sentiments=”labels”}
7. Import glove .6B dataset for word Vectors
8. Shuffle the dataset
9. Create word vectors
10. Create embedding matrix
11. Split the data into training and test set(70:30) ratio
12. train the RMDL model having (DNN CNN RNN as three internal models)
13. accuracy of trained model on testing dataset as accuracy ← model.evaluate()



5.TESTING

Testing is the process of evaluating a system or its component's with the intent to find that whether it satisfies the specified requirements or not .This activity results in the actual, expected and difference between their results i.e testing is executing a system in order to identify any gaps, errors or missing requirements in contrary to the actual desire or requirements.

5.1 Testing Strategies

In order to make sure that system does not have any errors, the different levels of testing strategies that are applied at different phases of software development are

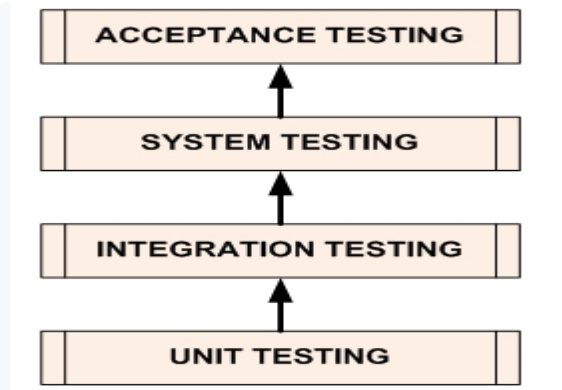


Figure 39: Phases of Software Development

5.1.1 Unit Testing

The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

5.1.2 Integration Testing

The testing of combined parts of an application to determine if they function correctly together is Integration testing .This testing can be done by using two different methods

5.1.2.1 Top Down Integration testing

In Top-Down integration testing, the highest-level modules are tested first and then progressively lower-level modules are tested.

5 .1.2.2 Bottom-up Integration testing

Testing can be performed starting from smallest and lowest level modules and proceeding one at a time .When bottom level modules are tested attention turns to those on the next level that use the lower level ones they are tested individually and then linked with the previously examined lower level modules.In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing.

5.1.3 System Testing

This is the next level in the testing and tests the system as a whole .Once all the components are integrated, the application as a whole is tested rigorously to see that it meets Quality Standards.

5.1.4 Acceptance Testing

The main purpose of this Testing is to find whether application meets the intended specifications and satisfies the client's requirements .We will follow two different methods in this testing.

5.1.4.1 Alpha Testing

This test is the first stage of testing and will be performed amongst the teams .Unit testing, integration testing and system testing when combined are known as alpha testing. During this phase, the following will be tested in the application:

- Spelling Mistakes.
- Broken Links.
- The Application will be tested on machines with the lowest specification to test loading times and any latency problems.

5.1.4.2 Beta Testing

In beta testing, a sample of the intended audience tests the application and send their feedback to the project team .Getting the feedback, the project team can fix the problems before releasing the software to the actual users.

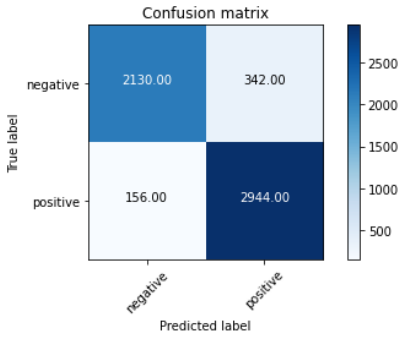
5.2 Test Results

The testing is done among the team members and by the end users. It satisfies the specified requirements and finally we obtained the results as expected.

6.Results

Some top Results among different model training are as follows:

- MCNN using Fasttext: The precision of the proposed model is approx. 91 percent. The recall of the proposed system is 91 percent. The accuracy achieved by the proposed system is up to 91 percent.



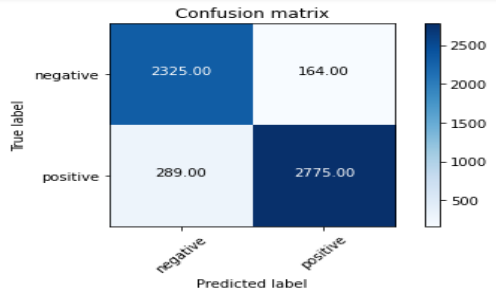

```
report = classification_report(y_test, predictions, target_names=['0','1'])
print(report)
```

```
              precision    recall  f1-score   support

           0       0.93      0.86      0.90      2472
           1       0.90      0.95      0.92      3100

    accuracy                           0.91      5572
   macro avg       0.91      0.91      0.91      5572
weighted avg       0.91      0.91      0.91      5572
```

- RNN using Fasttext: The precision of the proposed model is approx. 92 percent. The recall of the proposed system is 92 percent. The accuracy achieved by the proposed system is up to 92 percent.



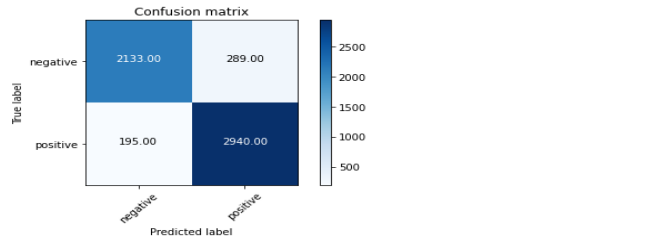

```
report = classification_report(y_test, predictions, target_names=['0','1'])
print(report)
```

```
              precision    recall  f1-score   support

           0       0.89      0.93      0.91      2489
           1       0.94      0.91      0.92      3064

    accuracy                           0.92      5553
   macro avg       0.92      0.92      0.92      5553
weighted avg       0.92      0.92      0.92      5553
```

- RNN using Word2Vec: The precision of the proposed model is approx. 91 percent. The recall of the proposed system is 91 percent. The accuracy achieved by the proposed system is up to 91 percent.



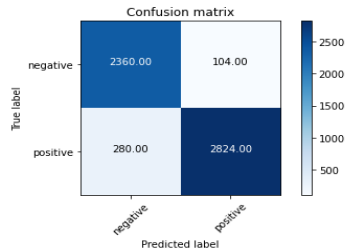

```
report = classification_report(y_test, predictions, target_names=['0','1'])
print(report)
```

```
              precision    recall  f1-score   support

           0       0.92      0.88      0.90      2422
           1       0.91      0.94      0.92      3135

    accuracy                           0.91      5557
   macro avg       0.91      0.91      0.91      5557
weighted avg       0.91      0.91      0.91      5557
```

- MCNN using Word2Vec: The precision of the proposed model is approx. 93 percent. The recall of the proposed system is 93 percent. The accuracy achieved by the proposed system is up to 93 percent.



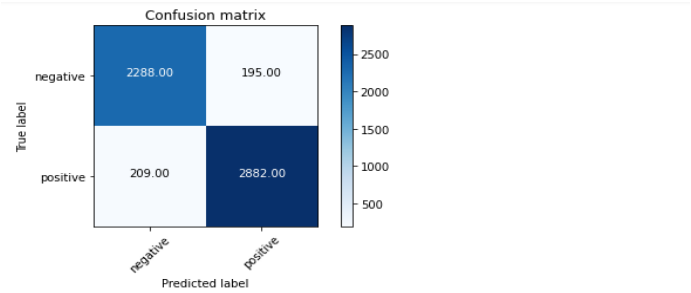

```
report = classification_report(y_test, predictions, target_names=['0','1'])
print(report)
```

```
              precision    recall  f1-score   support

           0       0.89      0.96      0.92      2464
           1       0.96      0.91      0.94      3104

    accuracy                           0.93      5568
   macro avg       0.93      0.93      0.93      5568
weighted avg       0.93      0.93      0.93      5568
```

- CNN using Fasttext: The precision of the proposed model is approx. 93 percent. The recall of the proposed system is 93 percent. The accuracy achieved by the proposed system is up to 93 percent.




```
report = classification_report(y_test, predictions, target_names=['0','1'])
print(report)
```

```
              precision    recall  f1-score   support

           0       0.92      0.92      0.92      2483
           1       0.94      0.93      0.93      3091

    accuracy                           0.93      5574
   macro avg       0.93      0.93      0.93      5574
weighted avg       0.93      0.93      0.93      5574
```

Table 1: A Comparative Study of different Deep Learning models using different Preprocessing Techniques

S.NO	Models used	Preprocessing Techniques	Accuracy
1	CNN	Word2vec	93.01
		Fasttext	92.59
2	MCNN	Word2Vec	93.10
		Fasttext	91.04
		BERT	92.36
3	RNN	Word2Vec	91.95
		Fasttext	92.19
4	HAN	Word2Vec	55.32
		Fasttext	55.42
		Glove	55.31
5	RMDL	BERT	91.22
		Fasttext	90.74
		Glove	91.05

- The best model for hotel review based Sentimental Analysis is MultiChannel Convolutional Neural Network using Word2Vec having an Accuracy of 93.10 Percent.

7.Conclusion

Various sentiment analysis methods and their level of analysing have been seen in this Project. Our ultimate aim is to come up with Sentiment Analysis which effectively calculate and categorize the user reviews. Research work is carried out for better analysis methods in this area, Here, different preprocessing techniques and different models are used to find best effective model among all for sentimental analysis. In the world of Internet majority of people depend on social networking sites to get their valued information, analysing the reviews from these projects will yield a better understanding and help in their decision-making.

8.References

1. Kumar, S. Sathish, and Aruchamy Rajini. "An Efficent Sentimental Analysis for Twitter Using Neural Network based on Rmsprop."

2. Bandara, P. M. A. U. *Analyze quality of products in e-commerce systems with sentimental analysis*. Diss. 2019.

3. Chen, Bingyang, Lulu Fan, and Xiaobao Fu. "Sentiment Classification of Tourism Based on Rules and LDA Topic Model." *2019 International Conference on Electronic Engineering and Informatics (EEI)*. IEEE, 2019.

4. Liu, Jason. “515K Hotel Reviews Data in Europe.” *Kaggle*, 21 Aug. 2017, www.kaggle.com/jiashenliu/515k-hotel-reviews-data-in-europe/discussion.

5. https://stackabuse.com/text-classification-with-bert-tokenizer-and-tf-2-0-in-python/

6. https://github.com/Hsankesara/DeepResearch/tree/master/Hierarchical_Attention_Network

7. https://www.kaggle.com/jiashenliu/515k-hotel-reviews-data-in-europe