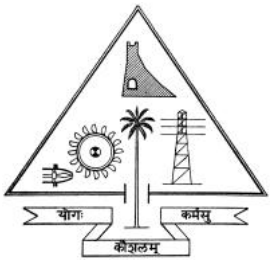
COMPARISON OF DEEP LEARNING TECHNIQUES FOR SENTIMENT ANALYSIS



CS 451 B.Tech Main Project Report 2019

Done By

Adithya M(TCR15CS002)

Harikrishnan R(TCR15CS024)

Jislin Varghese(TCR15CS028)

Saurav James Zacharias(TCR15CS052)

Guided By

Rahamathulla K

Assistant Professor

Dept of Computer Science and Engineering
Government Engineering College
Thrissur-680009

ABSTRACT

Sentiment analysis is understanding the polarity of textual data. It is used widely by companies to understand the customer sentiment to gain knowledge about market trends and product reception. It is a field which sees constant development and is still being extensively researched in different fields by leading organizations and universities all around the world.

In this project, we compare 6 deep learning techniques for their accuracy in Sentiment Analysis. We are comparing CNN, LSTM, CNN-LSTM, LSTM-CNN, ELMo and BERT.

Keywords: Sentiment analysis, CNN, LSTM, ELMo, BERT.

ACKNOWLEDGEMENT

We are extremely thankful to Prof. Rahmathulla K (Associate Professor, Dept. of CSE, Government Engineering College Thrissur) for guiding us in the making of this project. His insight and constant support has played a significant role in the making of this project.

We express my deep sense of gratitude to Mr. Ajay James (Associate Professor, Dept. of CSE, Government Engineering College Thrissur) for the encouragement and co-operation during the course of this project.

We are very thankful to Mrs. Helen K J (Assistant Professor, Dept. of CSE, Government Engineering College Thrissur) for her great support during the course of the project presentation.

We are grateful to Mr. Manoj Kumar K V (Head of the Department, Dept. of CSE, Government Engineering College Thrissur) for giving us the opportunity to work on this project.

We would also like to thank our parents who patiently helped us as we went through our work.

We also express my sense of gratitude to one and all that directly or indirectly helped us and assisted us in completing our work.

Last but not the least, the co-operation and help received from teachers and friends of Dept. of CSE, Government Engineering College Thrissur, is gratefully acknowledged.

Contents

List of Tables

List of Figures

List of abbreviations

ML Machine Learning
DL Deep Learning
NLP Natural Language Processing
CNN Convolution Neural Network
LSTM Long Short Term Memory
ELMo Bidirectional Language Model
BERT Bidirectional Encoder Representation for Transformer

Chapter 1

Introduction

Sentiment Analysis also known as Opinion Mining is a field within Natural Language Processing (NLP) that builds systems that try to identify and extract opinions within text. Usually, besides identifying the opinion, these systems extract attributes of the expression e.g.:

- Polarity: if the speaker express a positive or negative opinion,
- Subject: the thing that is being talked about,
- Opinion holder: the person, or entity that expresses the opinion.

Currently, sentiment analysis is a topic of great interest and development since it has many practical applications. Since publicly and privately available information over Internet is constantly growing, a large number of texts expressing opinions are available in review sites, forums, blogs, and social media.

With the help of sentiment analysis systems, this unstructured information could be automatically transformed into structured data of public opinions about products, services, brands, politics, or any topic that people can express opinions about. This data can be very useful for commercial applications like marketing analysis, public relations, product reviews, net promoter scoring, product feedback, and customer service.

There are many methods and algorithms to implement sentiment analysis systems, which can be classified as:

- Rule-based systems that perform sentiment analysis based on a set of manually crafted rules.
- Automatic systems that rely on machine learning techniques to learn from data.
- Hybrid systems that combine both rule based and automatic approaches.

Rule-based Approaches

Usually, rule-based approaches define a set of rules in some kind of scripting language that identify subjectivity, polarity, or the subject of an opinion. The rules may use a variety of inputs, such as the following:

- Classic NLP techniques like stemming, tokenization, part of speech tagging and parsing.
- Other resources, such as lexicons (i.e. lists of words and expressions).

This system is very nave since it doesn't take into account how words are combined in a sequence. A more advanced processing can be made, but these systems get very complex quickly. They can be very hard to maintain as new rules may be needed to add support for new expressions and vocabulary. Besides, adding new rules may have undesired outcomes as a result of the interaction with previous rules. As a result, these systems require important investments in manually tuning and maintaining the rules.

Automatic Approaches

Automatic methods, contrary to rule-based systems, don't rely on manually crafted rules, but on machine learning techniques. The sentiment analysis task is usually modeled as a classification problem where a classifier is fed with a text and returns the corresponding category, e.g. positive, negative, or neutral (in case polarity analysis is being performed).

Although we have sentiment analysis methods using machine learning techniques like naive bayes, regression tree,etc it cannot contextually understand the text. A sentence is treated as a bag of words and each word is treated separately without relation to words in its neighborhood.

Deep learning is prefered over traditional ML techniques as it can understand the context of the sentences and thus help in giving better sentiment analysis. Here we are using deep learning techniques such as convolutional neural network(CNN), Long-Short Term memory model and two networks that combine both convolutional and LSTM layers.

The issue with this approach is that it is sequential. The meaning of a word is dependent only on the words preceding it and not on those after. This makes long term dependency hard to learn and parallelism hard to implement. There is need for an approach that can evaluate the words bidirectionally.

ELMo is a deep contextualized word representation that models both (1) complex characteristics of word use (e.g., syntax and semantics), and (2) how these uses vary across linguistic contexts (i.e., to model polysemy). These word vectors are learned functions of the internal states of a deep bidirectional language model (biLM), which is pre-trained on a large text corpus. They can be easily added to existing models and significantly improve the state of the art across a broad range of challenging NLP problems, including question

answering, textual entailment and sentiment analysis.

Another approach is BERT, developed by GoogleAI. It is a naturally bidirectional pre-trained transformer which can be fine-tuned for any NLP task by adding additional layers and training on relevant data(labelled or unlabelled).

These approaches are new in the field and may replace the conventional techniques. There is need for a comparison of the performance of each.

As we have limited time and a minimal team, focus is on the proper collaboration of the individuals to produce a functional product swiftly. Our product is one with the scope for constant improvement. Considering these features, we choose **agile software development model** for our project.

The key reason for using agile model is the adaptability offered by the model. Since over project is a research on various techniques, there may be changes in design that occur during the development of the project. Agile model supports these changes and makes frequent testing and correction easy.

Chapter 2

Requirement Analysis

2.1 Requirement elicitation

For understanding the requirements of the project in depth, the following techniques were used:

- Literature Survey
- Brainstorming

2.1.1 Literature Survey

For this project, we refered to muliple papers dealing with sentiment analysis, word reprsentation techniques, data cleaning methods and most recent and relevant models for sentiment analysis. In this section, we discuss what we have learned from each paper.

Word2vec is a tool implementation of a training word vector model introduced by Google, which is made up of a simple three layers neural network. It uses the distributed representation proposed by [1] which can overcome the shortcomings of one-hot representation. The basic idea is: By the language model training, each word in a language is mapped into a fixed -length short vector, all of these vectors constitute a word vector space, each vector is regarded as a point in space, grammar, words is similar in grammar and semantic, so are their distance.

Paragraph Vector is an unsupervised framework that learns continuous distributed vector representations for pieces of texts. The texts can be of variable-length, ranging from sentences to documents. The name Paragraph Vector is to emphasize the fact that the method can be applied to variable-length pieces of texts, anything from a phrase or sentence to a large document[2]. The vector representation is trained to be useful for predicting words in a paragraph. More precisely, we concatenate the paragraph vector with several word vectors from a paragraph and predict the following word in the given context. Both word vectors and paragraph vectors are trained by the stochastic gradient descent

and backpropagation [3][4][5]. In their formulation, each word is represented by a vector which is concatenated or averaged with other word vectors in a context, and the resulting vector is used to predict other words in the context. For example, the neural network language model proposed in [6] uses the concatenation of several previous word vectors to form the input of a neural network, and tries to predict the next word. The outcome is that after the model is trained, the word vectors are mapped into a vector space such that Distributed Representations of Sentences and Documents semantically similar words have similar vector representations.

Document level sentiment classification is a fundamental task in sentiment analysis, and is crucial to understand user generated content in social networks or product reviews [7][8][9]. The task calls for identifying the overall sentiment polarity (e.g. thumbs up or thumbs down, 1-5 stars on review sites) of a document[10]. In literature, dominant approaches follow[11] and exploit machine learning algorithm to build sentiment classifier. Many of them focus on designing hand-crafted features [12] or learning discriminate features from data, since the performance of a machine learner is heavily dependent on the choice of data representation [13].

Most previous research on sentiment-based classification has been at least partially knowledge-based. Some of this work focuses on classifying the semantic orientation of individual words or phrases, using linguistic heuristics or a pre-selected set of seed words [14][15]. Past work on sentiment-based categorization of entire documents has often involved either the use of models inspired by cognitive linguistics [16] or the manual or semi-manual construction of discriminant-word lexicons [17] [18].

Convolutional neural networks are deep artificial neural networks that are used primarily to classify images (e.g. name what they see), cluster them by similarity (photo search), and perform object recognition within scenes. They are algorithms that can identify faces, individuals, street signs, tumors, platypuses and many other aspects of visual data.

CNN is a class of deep, feed-forward artificial neural networks (where connections between nodes do not form a cycle) use a variation of multilayer perceptrons designed to require minimal preprocessing. These are inspired by animal visual cortex.

CNNs are generally used in computer vision, however theyve recently been applied to various NLP tasks and the results were promising

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture[1] used in the field of deep learning. Unlike standard feed forward neural networks, LSTM has feedback connections that make it a "general purpose computer". It can not only process single data points (such as images), but also entire sequences of data (such as speech or video).

For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.

Experimental results [19] show that CNN-based text classification method is more accurate than the best method at that time. Thanks to and [20] CNN -based text classification methods have been improved in special scenarios.

The model proposed in [21] is to feed input features, surrounded by temporal context, into a few CNN layers to reduce spectral variation. The output of the CNN layer is then fed into a few LSTM layers to reduce temporal variations. Then, the output of the last LSTM layer is fed to a few fully connected DNN layers, which transform the features into a space that makes that output easier to classify.

For ensemble learning [22] a method is described that integrates the posterior probabilities produced by different deep learning systems using the framework of stacking as a class of techniques for forming combinations of different predictors to give improved prediction accuracy. It compares the accuracy of DNN,CNN,RNN along with their combinations.

Before popular word embedding techniques like word2vec around 2013, NLP didn't really have nice ways to represent word semantics in a continuous vector space. People used the bag of words (BoW) representation, which is simply a way to map each unique token to a dimension (an axis) in the N-dimensional space by ignoring the word order completely. Clearly, BoW has several issues, one of which is its inability to represent semantic similarity (or dissimilarity) between words.

Word embeddings solve this exact issue by representing words not just by one-hot vectors but by sets of continuous numbers. This is why the use of word embeddings has become so popular in recent years in NLP. Unlike traditional word embedding methods, ELMo is dynamic, meaning that ELMo embeddings change depending on the context even when the word is the same.

In March 2018, Allen Institute for Artificial Intelligence released a paper on ELMo, a bi-directional approach to NLP. [23]

Pre-trained word representations [24] [25] are a key component in many neural language understanding models. However, learning high quality representations can be challenging. They should ideally model both (1) complex characteristics of word use (e.g., syntax and semantics), and (2) how these uses vary across linguistic contexts (i.e., to model polysemy).

There have been a few attempts to leaning context-dependent representations. Context2Vec [26] uses a bidirectional Long Short Term Memory [27] to encode the context around a pivot word. Other approaches for learning contextual embeddings include the pivot word itself in the representation and are computed with the encoder of either a supervised neural machine transla-

tion (MT) system [28] or an unsupervised language model [29]. Both of these approaches benefit from large datasets, although the MT approach is limited by the size of parallel corpora. In this paper, we take full advantage of access to plentiful monolingual data, and train our biLM on a corpus with approximately 30 million sentences [30].

ELMo introduces a new type of deep contextualized word representation that directly addresses both challenges, can be easily integrated into existing models, and significantly improves the state of the art in every considered case across a range of challenging language understanding problems.

ELMo uses vectors derived from a bidirectional LSTM that is trained with a coupled language model (LM) objective on a large text corpus. For this reason, we call them ELMo (Embeddings from Language Models) representations. Unlike previous approaches for learning contextualized word vectors [31][32], ELMo representations are deep, in the sense that they are a function of all of the internal layers of the biLM. More specifically, it learns a linear combination of the vectors stacked above each input word for each end task, which markedly improves performance over just using the top LSTM layer.

BERT stands for Bidirectional Encoder Representations from Transformers. BERT is designed to pre-train deep bidirectional representations by jointly conditioning on both left and right context in all layers. BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE benchmark to 80.4% (7.6% absolute improvement), MultiNLI accuracy to 86.7% (5.6% absolute improvement) and the SQuAD v1.1 question answering Test F1 to 93.2 (1.5 absolute improvement), outperforming human performance by 2.0.[33]Language model pre-training has shown to be effective for improving many natural language processing tasks[34].

Bidirectional Encoder Representations from Transformers.BERT addresses the previously mentioned unidirectional constraints by proposing a new pre-training objective: the masked language model (MLM). The masked language model randomly masks some of the tokens from the input, and the objective is to predict the original vocabulary id of the masked word based only on its context. Unlike left-to-right language model pre-training, the MLM objective allows the representation to fuse the left and the right context, which allows us to pre-train a deep bidirectional Transformer.

BERT uses masked language models to enable pre-trained deep bidirectional representations, unlike [35], which uses unidirectional language models for pretraining. This is also in contrast to [36], which uses a shallow concatenation of independently trained left to-right and right-to-left LMs.

BERT is the first fine-tuning based representation model that achieves state-of-the-art performance on a large suite of sentence-level and token-level

tasks, outperforming many systems with task-specific architectures. There is a long history of pre-training general language representations, and we briefly review the most popular approaches in this section.

Feature-based Approaches: Pretrained word embeddings are considered to be an integral part of modern NLP systems, offering significant improvements over embeddings learned from scratch. ELMo [37] generalizes traditional word embedding research along a different dimension. They propose to extract context sensitive features from a language model. When integrating contextual word embeddings with existing task-specific architectures, ELMo advances the state-of-the-art for several major NLP benchmarks including question answering on SQuAD, sentiment analysis , and named entity recognition.

Fine Tuning Approaches: A recent trend in transfer learning from language models (LMs) is to pre-train some model architecture on a LM objective before fine-tuning that same model for a supervised downstream task[38]. The advantage of these approaches is that few parameters need to be learned from scratch. At least partly due this advantage, OpenAI GPT achieved previously state-of-the-art results on many sentence level tasks from the GLUE (General Language Understanding Evaluation) benchmark. BERTs model architecture is a multi-layer bidirectional Transformer encoder based on the original implementation described in [39].

We researched on these papers and found the need for comparing deep learning models used for sentiment analysis. In our report, we aim to assess the performance of all these models on the same dataset and draw evaluations from the same.

2.2 Functional Requirements

An overview of the series of steps or functions that need to be carried out in order to obtain the final result are enumerated below. Note that these maybe not be the only functions that might be performed in the actual execution phase. Depending on run-time scenarios, more steps maybe added. More number of steps may also be added if the project develops further into more ideas.

- Import required models and datasets.
- Divide dataset into train set(80%)and test set(20%). We will train and validate the model using cross-validation by applying on varying divisions of train set used as validation set. Final test set is used to compare performance measures for the models.
- Perform data cleaning tasks on the dataset.
- Assign unique indices for the tokens

- Train the 4 DNNs, ELMo and BERT
- Compare performance of each model using accuracy, precision, recall, F1 score measure and time complexity in the testing and training phase.
- Perform data visualisation techniques like wordcloud to further understand the data.

2.3 User Characteristics

The characteristics necessary for a potential user of the proposed algorithms are described below.

The end result of the project is particularly aimed at an audience of DL researchers and developers. It is quite essential that they understand the basics of DL in order to utilise these models. Knowledge of word ebedding techniques that preserve the context is of importance. If new technologies develop eventually, they can compare their results to the results obtained in this report. That would increase the audience and requirements but basic need is understanding of deep learning.

2.4 Requirements Validation

Prior to subsequent phases, the specified requirements need to be checked to ascertain that they align with the elicited requirements. This has been carried out at the primary level by conducting careful scrutiny of the following factors:-

- Necessity: It has been understood from the literature survey conducted that word embedding is of significant importance in the upcoming NLP related research works, which has been cited as the primary requirement of this project.
- Correctness and Unambiguity: The specified requirements describes the desired functionalities and have been found to be technically correct as well.
- Technical Availability: The implementation of the requirements are technically available and also feasible under the current circumstances.

The aforementioned requirements of the proposed project have further been validated using a checklist based reader technique. The requirements were examined by two peers to discover defects, with the help of a checklist so that the knowledge of the individual conducting the walkthrough was not reflected in its outcome. The checklist had the following questions:-

- Has the topic been introduced clearly?
- Has the relevant information been extracted from the research papers?

- Are the observations drawn logical and pertinent?
- Have the requirements been clearly worded?
- Is there any ambiguity in the requirements?
- Does the implementation of requirements seem to be impossible?

The outcome of this walk-through was fairly positive and required changes were made in the document that addressed the issues pointed out. Thus, the specified requirements have been validated against the elicited requirements.

2.5 Change Management Process

Any changes in the above mentioned specifications will arise at the coding and testing phase of the project. Since the project has been proposed to make use of an agile model, it is easier to iterate over the phases as many times as required until we evaluate all algorithms to arrive at the most efficient one. Thus, in case of an additional requirement that might come up during the course of the project can be accommodated quite effectively without affecting the project schedule considerably.

But on the other hand, the requirements of any project that is a part of a larger research area which is growing exponentially in the current circumstances will undergo changes as related research progresses. There is an available user group of researchers, but their feedback comes only very later in the project, when they actually decide to use the algorithm in their research work. That might take up years and hence any modifications shall happen only when the team is together in the developing phase of the end result. Implementing feedback is way into the future and unpredictable.

Thus, while the immediate requirement changes that need to be implemented can be accommodated without much difficulty, the changes in requirements that may come about far into the future are unpredictable and hence cannot be resolved within the duration of the project.

Chapter 3

Design

This section describes the design patterns that is going to be used in the project. It presents a high level view of the system and the technologies that should be used for the successful implementation of the project.

3.1 User Interface

The user can view the result of the analysis on the Colab Notebook of Google, which is a cloud based notebook for DL/ML development. It comes with GPU support and offers 12 GB of RAM. You can see the real-time result in this notebook. The code is executed in units. In the notebook there is an area for the text, code and for showing the output.

3.2 Models

3.2.1 CNN

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. CNNs use a variation of multilayer perceptron designed to require minimal preprocessing. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

Sentence: Here there are 7 words in the given sentence including exclamation mark. The dimension of the words is 5.

Filters: Three different filters are used in this case.

Feature maps: The action of two filter word is shown below. Each elements of the filter is multiplied with the sentence matrix and the result is (0.6 x 0.2 + 0.5 x 0.1 + + 0.1 x 0.1 = 0.51) 0.51. Similarly it moves down and the next rows of the sentence matrix is multiplied.

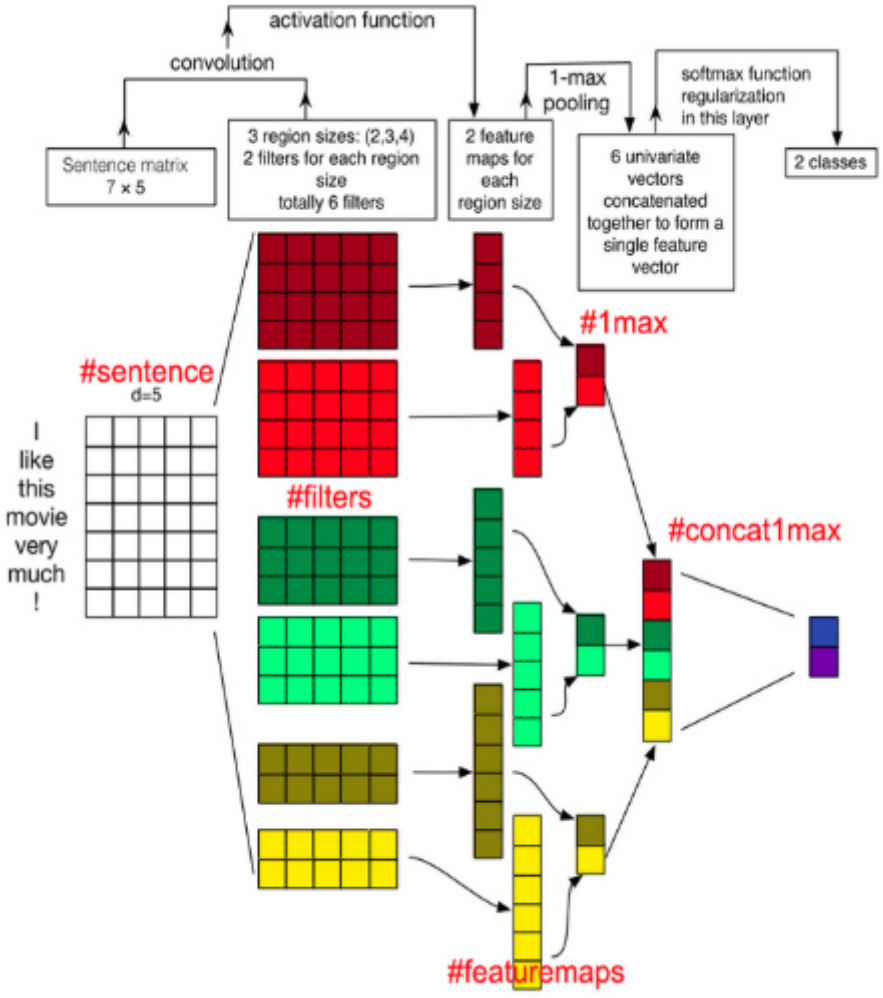


Figure 3.1: CNN Model

1max: Here the largest number from each vector is selected.

concat1max: After performing the pooling a fixed size vector of 6 elements is obtained. This vector is fed into a softmax layer and the sentence is classified into two classes, positive and negative sentiment. The errors are propagated back into the network.

3.2.2 LSTM

Long Short Term Memory network (LSTM) is a special type of RNN, which is capable of learning long-term dependencies. All RNNs have the form of a chain of repeating modules. In standard RNNs, this repeating module normally has a simple structure. However, the repeating module for LSTM is more complicated. Instead of having a single neural network layer, there are four layers interacting in a special way. Besides, it has two states: hidden state and cell state.

LSTMs are Recurrent Neural Networks composed of LSTM(Long Short

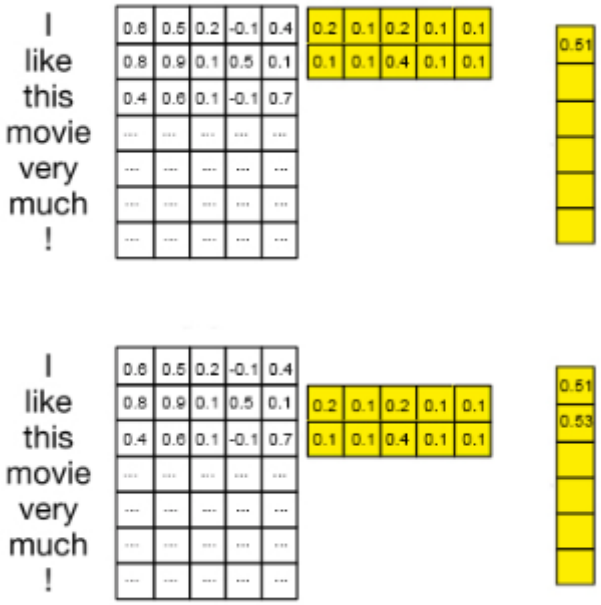


Figure 3.2: CNN Pooling

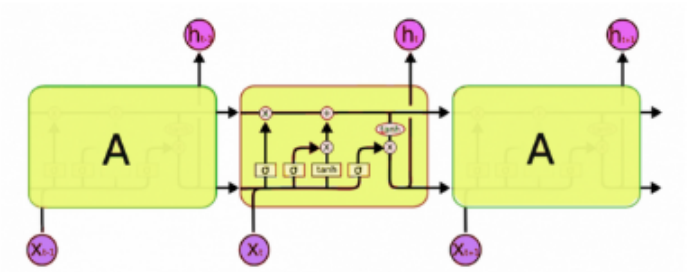


Figure 3.3: LSTM

Term Memory) units. Recurrent Neural Networks suffers from many drawbacks such as not being able to remember the context of the sentence for a long time, and not having the ability to selectively forget. These problems are not present in LSTMs since they have the ability to remember for a long time as well as to selectively forget. LSTMs mainly consists of three gates:

Input Gate: This gate stores the input as a vector while forgetting unnecessary details. Sigmoid function is responsible for forgetting the information. It produces values between 0 and 1. If the value is 1, it remembers and if the value is 0, it forgets it. tanh is used to create input vectors.

Forget Gate: This gate is used for selectively forgetting information. Its done with the help of sigmoid function.

Output Gate: This gate produces the output of the LSTM network. The functioning of an output gate can again be broken down to three steps:(1)Creating

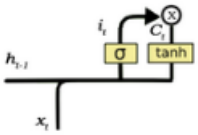


Figure 3.4: LSTM Input Gate

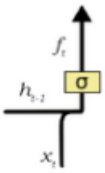


Figure 3.5: LSTM Forget Gate

a vector after applying tanh function to the cell state, thereby scaling the values to the range -1 to +1. (2)Making a filter using the values of h_{t-1} and x_t, such that it can regulate the values that need to be output from the vector created above. This filter again employs a sigmoid function. (3)Multiplying the value of this regulatory filter to the vector created in step 1, and sending it out as a output and also to the hidden state of the next cell.

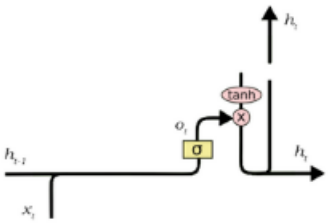


Figure 3.6: LSTM Output Gate

3.2.3 CNN-LSTM

The CNN LSTM architecture involves using Convolutional Neural Network (CNN) layers for feature extraction on input data combined with LSTMs to support sequence prediction. In CNN-LSTM Model, the first layer is a Convolution Layer which receives word as inputs. Followed by convolution, the output is pooled by Max Pooling. After that the output is fed into the LSTM layer. Finally the input textual data is classified as positive sentiment or negative sentiment.

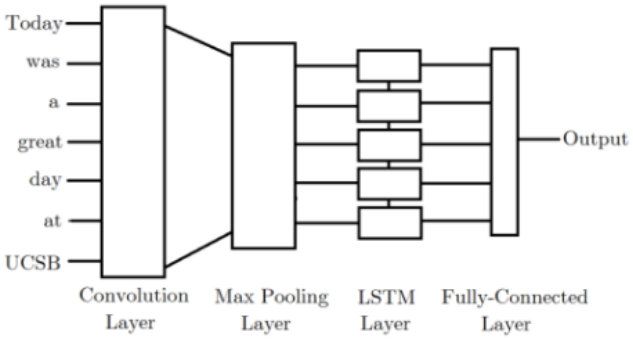


Figure 3.7: CNN-LSTM

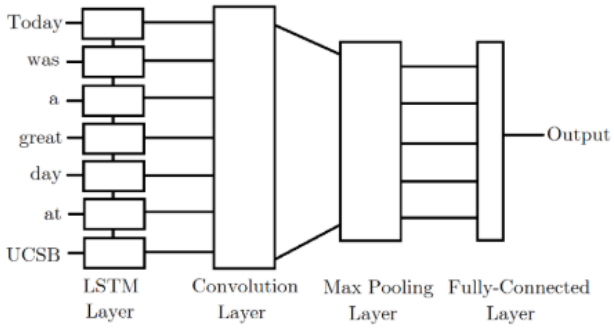


Figure 3.8: LSTM-CNN

3.2.4 LSTM-CNN

In the LSTM-CNN Model the first layer is the LSTM layer and the next layer is the CNN layer. The input is fed to the LSTM layer. The output from the LSTM layer is fed to the CNN layer, in which convolution occurs and output is pooled and ultimately outputted as either positive or negative sentiment.

3.2.5 ELMo

ELMo is a deep contextualized word representation that models both (1) complex characteristics of word use (e.g., syntax and semantics), and (2) how these uses vary across linguistic contexts (i.e., to model polysemy). These word vectors are learned functions of the internal states of a deep bidirectional language model (biLM), which is pre-trained on a large text corpus. They can be easily added to existing models and significantly improve the state of the art across a broad range of challenging NLP problems, including question answering, textual entailment and sentiment analysis.

The pre-trained biLMs in ELMo are similar to the architectures in [39] and [40], but modified to support joint training of both directions and add a residual connection between LSTM layers. We focus on large scale biLMs in this work, as [41] highlighted the importance of using biLMs over forward-only

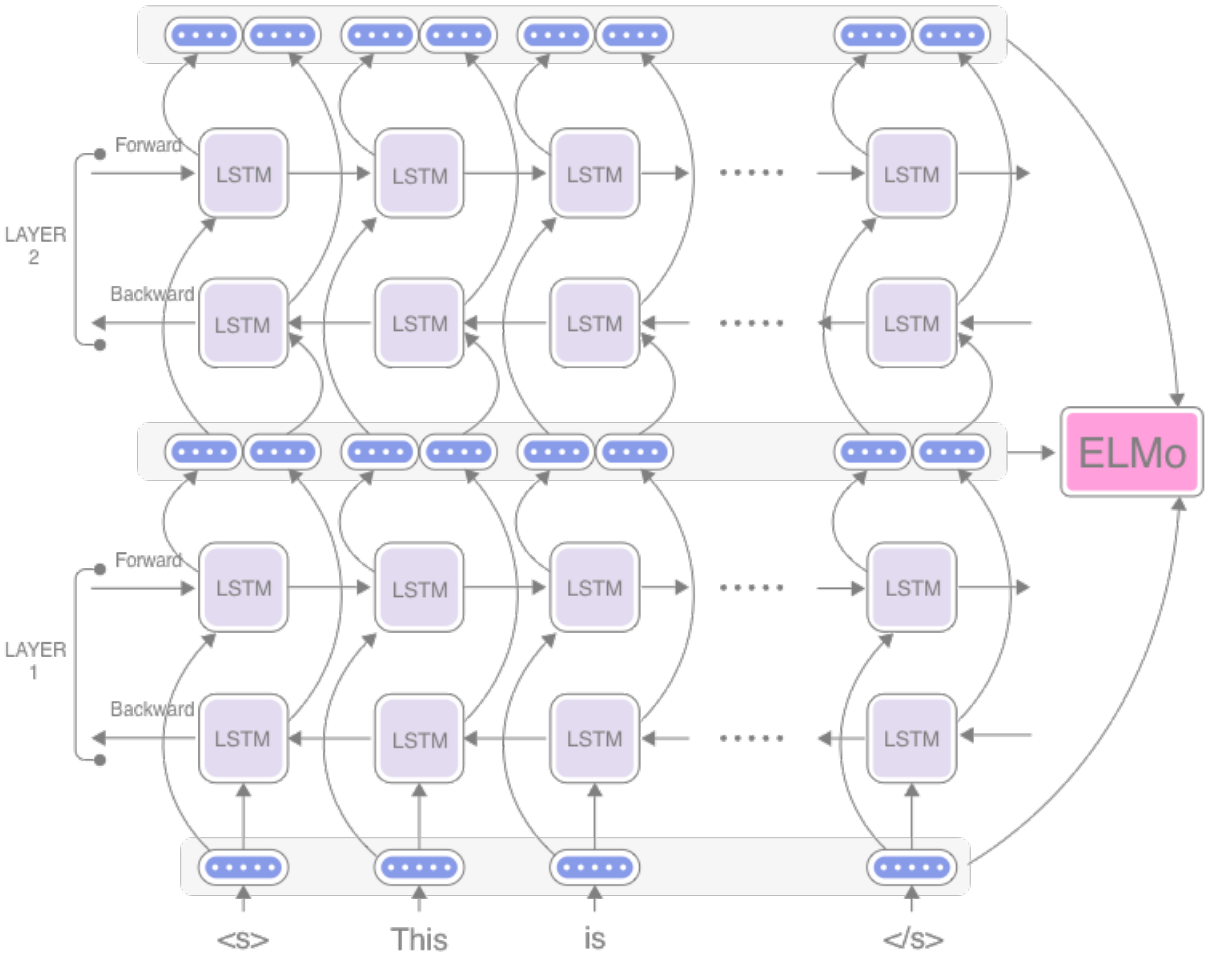


Figure 3.9: ELMo

LMs and large scale training.

The final model uses L= 2 biLSTM layers with 4096 units and 512 dimension projections and a residual connection from the first to second layer. The context insensitive type representation uses 2048 character n-gram convolutional filters followed by two highway layers [42] and a linear projection down to a 512 representation. As a result, the biLM provides three layers of representations for each input token, including those outside the training set due to the purely character input. In contrast, traditional word embedding methods only provide one layer of representation for tokens in a fixed vocabulary.

3.2.6 BERT

BERT stands for Bidirectional Encoder Representations from Transformers. The BERT models architecture is a bidirectional Transformer encoder.The use of a Transformer comes as no surprisethis is a recent trend due Transformers training efficiency and superior performance in capturing long-distance dependencies compared to a recurrent neural network architecture. The bidirectional encoder meanwhile is a standout feature that differentiates BERT from OpenAI GPT (a left-to-right Transformer) and ELMo (a concatenation of independently trained left-to-right and right- to-left LSTM).

Most competitive neural sequence transduction models have an encoder-decoder structure, [43][44]. Here, the encoder maps an input sequence of symbol representations (x1; :::; xn) to a sequence of continuous representations z = (z1; :::; zn). Given z, the decoder then generates an output sequence (y1; :::; ym) of symbols one element at a time. At each step the model is auto-regressive [45], consuming the previously generated symbols as additional input when generating the next. The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of figure, respectively.

3.2.6.1 Encoder and decoder

Encoder: The encoder is composed of a stack of N = 6 identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position wise fully connected feed-forward network. We employ a residual connection around each of the two sub-layers, followed by layer normalization. Decoder: The decoder is also composed of a stack of N = 6 identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

3.2.6.2 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key. For "Scaled Dot-Product Attention" (Figure 3.12), the input consists of queries and keys of dimension dk, and values of dimension dv. We compute the dot products of the query with all keys, divide each by pdk, and apply a softmax function to obtain the weights on the values.

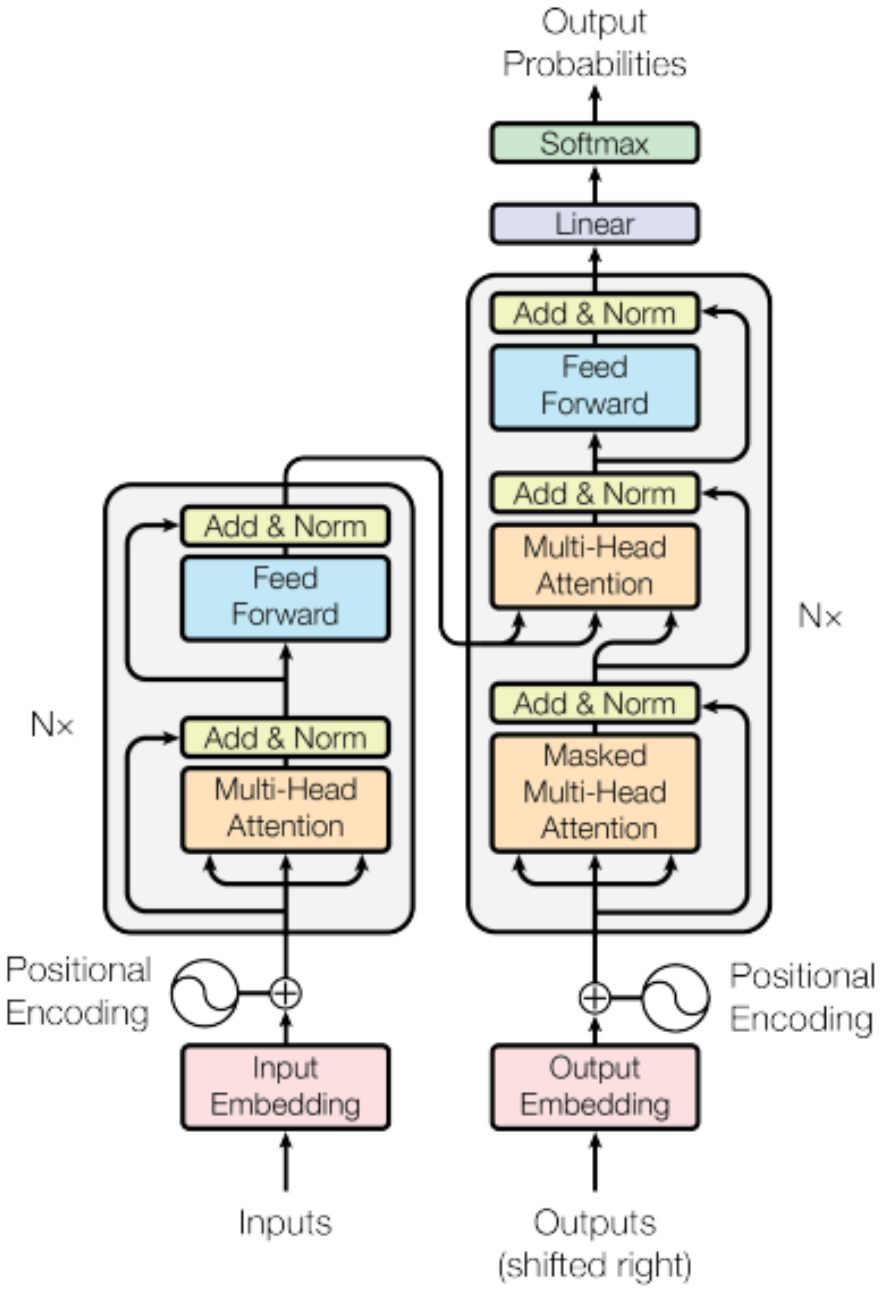


Figure 3.10: The Transformer - model architecture

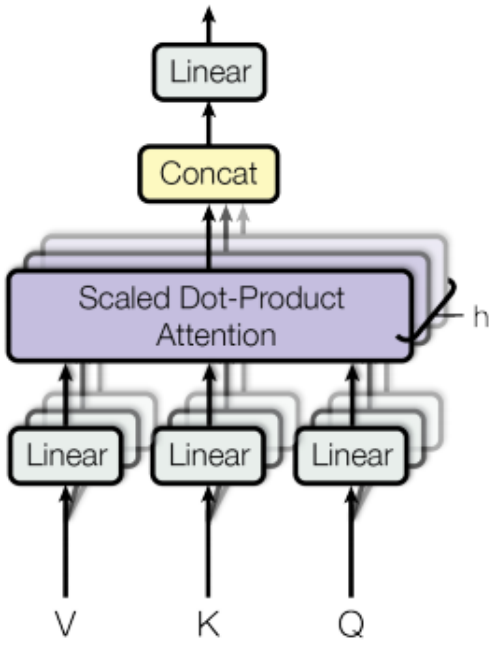


Figure 3.11: Multi-Head Attention.

3.3 Algorithm

The following is the algorithm used for selecting the best sentiment analysis model.

1. Start.
2. Import required libraries like pandas, keras, numpy,etc
3. Import the dataset.
4. Convert 1-5 rating into 0 and 1 values and save into a new dataset as labels Prepossessing.
5. Make all characters lower-case.
6. Get rid of apostrophes punctuation, commas and numbers
7. Remove stop-words (irrelevant words that occur in large numbers Eg: is, was, an, a, etc)
8. Convert words to their base form (Eg: eating -eat)
9. Tokenise each word and make a dictionary with each word having a unique index
10. Convert each sentence into a sequence of indices

11. Divide dataset into 2, with 80% and 20% data respectively.
12. Train the following models on the larger dataset using cross-validation:
    - LSTM
    - LSTM-CNN
    - CNN-LSTM
    - CNN
    - ELMo
    - Bert
13. Test each model on the remaining dataset
14. Compare the accuracy, precision, recall, and f1 measure of all models and select the best model.
15. Analyse the examples that are misclassified to further improve the system
16. Stop.

3.4 Test Case Design

We use 80 percentage of the Amazon food review dataset for training the networks. After training the network, we evaluate their performance by considering accuracy, recall, precision, f1 measures and time complexity of the models. During the training of these models cross validation is also done. This can be done using methods such as K-Cross Validation.

Chapter 4

Coding

4.1 Loading of Data

One of the very first step we have to do in Sentiment analysis is loading of the dataset. We are using Colab notebooks created by Google for doing our project. This was created as part of Google research project created to help disseminate machine learning education and research. There are three ways to upload dataset into Colab,

1. Through GitHUb
2. Through local drive
3. Through Google drive

We have found that uploading the dataset using Google drive is the most effective method. The dataset, which is Amazon food reviews consisting of approximately 5 lakh dataset is first uploaded to the Google drive and from there its uploaded to Colab. The code for uploading the dataset from Colab is shown below.

```
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

It requires the authentication from Google drive before the dataset can be uploaded to Colab. The dataset is uploaded to Google drive via PyDrive.

4.2 Preprocessing

From the ratings given in the food review we classify each review into positive and negative. The ratings 1 to 3 is classified as negative sentiment, which is given a 0 and ratings from 4 to 5 is classified as positive sentiment, which is given 1. After this the number of positive and negative reviews are balanced. For balancing, get the number of positive and negative reviews. Then whichever is the smallest value make the other the same value by taking random number of samples from the largest equal to the smallest value.

```
# Change from 1-5 ratings to negative or positive sentiment
rating_to_sentiment = { 1: 0, 2: 0, 3: 0, 4: 1, 5: 1 }
data['Sentiment'] = data['Score'].apply(lambda x:
    rating_to_sentiment[x])

# Count number of negative and positive reviews
neg_num = pd.value_counts(data['Sentiment'])[0]
pos_num = pd.value_counts(data['Sentiment'])[1]

print('# negative reviews before: {}'.format(neg_num))
print('# positive reviews before: {}'.format(pos_num))

# Make the data set balanced
balanced_sample_num = np.min([neg_num, pos_num])

# Picks <'balanced_sample_num'> numbers of negative and positive
    reviews at random
data = (data.groupby('Sentiment', as_index = False)
        .apply(lambda x: x.sample(n = balanced_sample_num))
        .reset_index(drop = True))
```

Since we are using textual data written by users it will contain a lot of noises . These need to be cleaned before they can be analysed further for optimal results. In data cleaning we are processing the following types of data

1. Escaping html characters: Data obtained from web usually contains a lot of html entities. It is thus necessary to get rid of these entities.

2. Removal of Stopwords: When data analysis needs to be data driven at the word level, the commonly occurring words (stop-words) should be removed.

3. Find the base form of the word: Verb can be in different tenses. By finding the root form of the verb we can improve the accuracy in sentiment analysis.

4. Tokenization and Vectorization: In tokenization we convert the sentences into tokens and then they are converted to vector representation by vectorization.

```
# All characters to lower case
data['Text'] = data['Text'].apply(lambda x: x.lower())

# Remove html-tags, punctuation, commas, numbers etc
data['Text'] = data['Text'].apply((lambda x: re.sub('<[^<]+?>', ' ',
    x)))
data['Text'] = data['Text'].apply((lambda x:
    re.sub('[^a-zA-z0-9\s]', ' ', x)))
data['Text'] = data['Text'].apply((lambda x:
    re.sub('^\d+\s|\s\d+\s|\s\d+$', ' ', x)))

# Convert text into tokens, in this case sentences into words
data['Text'] = data.apply(lambda x: word_tokenize(x['Text']), axis =
    1)

# Remove most commonly occuring words which are not relevant in the
    context of the data
irrelevant_words = stopwords.words('english')
data['Text'] = data['Text'].apply(lambda x: [word for word in x if
    word not in irrelevant_words])

# Find the base form of the word (lemmatization)
lemma = WordNetLemmatizer()
data['Text'] = data['Text'].apply(lambda x: "
    ".join([lemma.lemmatize(word) for word in x]))
print(data['Text'])
# Vectorize the text by turning each review into a sequence of
    integers
#(each integer being theindex of a token in a dictionary)
# Also, pad so that every review has the same length
num_top_words = 10000
tokenizer = Tokenizer(num_words = num_top_words, split = ' ')
tokenizer.fit_on_texts(data['Text'].values)
X = tokenizer.texts_to_sequences(data['Text'].values)
X = pad_sequences(X)
```

We are comparing the performance of 6 models. They include CNN, LSTM, CNN-LSTM, LSTM-CNN, ELMo, and BERT. The implementation of these models are shown below.

1. **LSTM**

```
def lstm_model(num_top_words, input_length):
    model = Sequential()
    model.add(Embedding(input_dim = num_top_words, output_dim =
        128, input_length = input_length))
    model.add(CuDNNLSTM(100))
    model.add(Dense(2, activation = 'softmax'))
    model.compile(loss = 'categorical_crossentropy', optimizer
        = 'adam', metrics = ['accuracy'])
```

```
    return model
```

2. **CNN**

```
def cnn_model(num_top_words, input_length):
    model = Sequential()
    model.add(Embedding(input_dim = num_top_words, output_dim =
        128, input_length = input_length))
    model.add(Dropout(0.5))
    model.add(Conv1D(128, kernel_size = 10, input_shape =
        (input_length, num_top_words), activation = 'relu',
        kernel_regularizer=regularizers.l2(0.01)))
    model.add(MaxPooling1D())
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(2, activation = 'softmax',
        kernel_regularizer = regularizers.l2(0.01)))
    model.compile(loss = 'categorical_crossentropy', optimizer
        = 'adam', metrics = ['accuracy'])
    return model
```

3. **CNN-LSTM**

```
def cnn_lstm_model(num_top_words, input_length):
    model = Sequential()
    model.add(Embedding(input_dim = num_top_words, output_dim =
        128, input_length = input_length))
    model.add(Conv1D(128, kernel_size = 10, input_shape =
        (input_length, num_top_words), activation = 'relu'))
    model.add(MaxPooling1D())
    model.add(Dropout(0.5))
    model.add(CuDNNLSTM(100))
    model.add(Dense(2, activation = 'softmax'))
    model.compile(loss = 'categorical_crossentropy', optimizer
        = 'adam', metrics = ['accuracy'])
    return model
```

4. **LSTM-CNN**

```
def lstm_cnn_model(num_top_words, input_length):
    model = Sequential()
    model.add(Embedding(input_dim = num_top_words, output_dim =
        128, input_length = input_length))
    model.add(CuDNNLSTM(100, return_sequences = True))
    model.add(Conv1D(128, kernel_size = 10, activation =
        'relu'))
    model.add(MaxPooling1D())
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(2, activation = 'softmax'))
    model.compile(loss = 'categorical_crossentropy', optimizer
        = 'adam', metrics = ['accuracy'])
    return model
```

5. **Custom Layer Built for ELMo**

```
class ElmoEmbeddingLayer(Layer):
    def __init__(self, **kwargs):
        self.dimensions = 1024
        self.trainable=True
        super(ElmoEmbeddingLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        self.elmo =
            hub.Module('https://tfhub.dev/google/elmo/2',
            trainable=self.trainable,
                                name="{}_module".format(self.name))

        self.trainable_weights +=
            K.tf.trainable_variables(scope="^{}_module/.*".format(self.name))
        super(ElmoEmbeddingLayer, self).build(input_shape)

    def call(self, x, mask=None):
        result = self.elmo(K.squeeze(K.cast(x, tf.string),
            axis=1),
                    as_dict=True,
                    signature='default',
                    )['default']
        return result

    def compute_mask(self, inputs, mask=None):
        return K.not_equal(inputs, '--PAD--')

    def compute_output_shape(self, input_shape):
        return (input_shape[0], self.dimensions)
```

6. **Function to Build ELMo**

```
!pip install keras_metrics
import keras
import keras_metrics
def build_model():
  input_text = layers.Input(shape=(1,), dtype="string")
  embedding = ElmoEmbeddingLayer()(input_text)
  dense = layers.Dense(256, activation='relu')(embedding)
  pred = layers.Dense(1, activation='sigmoid')(dense)

  model = Model(inputs=[input_text], outputs=pred)

  model.compile(loss='binary_crossentropy', optimizer='adam',
      metrics=['accuracy'])


  model.summary()
  #,keras_metrics.precision(), keras_metrics.recall()
  return model
```

7. **BERT Model**

```
def create_model(is_predicting, input_ids, input_mask,
    segment_ids, labels,
                 num_labels):
  """Creates a classification model."""

  bert_module = hub.Module(
      BERT_MODEL_HUB,
      trainable=True)
  bert_inputs = dict(
      input_ids=input_ids,
      input_mask=input_mask,
      segment_ids=segment_ids)
  bert_outputs = bert_module(
      inputs=bert_inputs,
      signature="tokens",
      as_dict=True)

  # Use "pooled_output" for classification tasks on an entire
      sentence.
  # Use "sequence_outputs" for token-level output.
  output_layer = bert_outputs["pooled_output"]

  hidden_size = output_layer.shape[-1].value

  # Create our own layer to tune for politeness data.
  output_weights = tf.get_variable(
      "output_weights", [num_labels, hidden_size],
```

```
    initializer=tf.truncated_normal_initializer(stddev=0.02))

output_bias = tf.get_variable(
    "output_bias", [num_labels],
       initializer=tf.zeros_initializer())

with tf.variable_scope("loss"):

  # Dropout helps prevent overfitting
  output_layer = tf.nn.dropout(output_layer, keep_prob=0.9)

  logits = tf.matmul(output_layer, output_weights,
      transpose_b=True)
  logits = tf.nn.bias_add(logits, output_bias)
  log_probs = tf.nn.log_softmax(logits, axis=-1)

  # Convert labels into one-hot encoding
  one_hot_labels = tf.one_hot(labels, depth=num_labels,
      dtype=tf.float32)

  predicted_labels = tf.squeeze(tf.argmax(log_probs, axis=-1,
      output_type=tf.int32))
  # If we're predicting, we want predicted labels and the
      probabilties.
  if is_predicting:
    return (predicted_labels, log_probs)

  # If we're train/eval, compute loss between predicted and
      actual label
  per_example_loss = -tf.reduce_sum(one_hot_labels *
      log_probs, axis=-1)
  loss = tf.reduce_mean(per_example_loss)
  return (loss, predicted_labels, log_probs)
```

Chapter 5

Results

In this section, we discuss the results of the research work we did. Our task was to the find the best among CNN, LSTM, CNN-LSTM, LSTM-CNN, BERT, and ELMO. We used the metrics Accuracy, Precision, Recall and F1 score for determining the best model since these parameters are widely used for comparing a models efficiency and how good it is in classification.

Each of these parameters have its own advantages and disadvantages. Accuracy is the ratio of observations that were correctly labelled by the system to the total number of observations. But it cant be relied all the time. In class imbalance problem, where the main class of interest is less, accuracy measures are often misleading. Example, there is a model for cancer detection with 97% accuracy. But what if the 3% of the dataset actually had cancer. Then the model failed completely in this case.

Precision is a measure of how much of the positively labeled values are correct. This is an important parameter when the cost of false positive is high. Example in a situation where a mail is labelled falsely as spam. This can result in a person missing important mail.

Recall is the ratio of true positive to actual true positive. This helpful in testing models where the cost associated with false negative is high. For instance, in fraud detection or sick patient detection. If a fraudulent transaction (Actual Positive) is predicted as non-fraudulent (Predicted Negative), the consequence can be very bad for the bank. F1 Score is useful in situations where we need a balance between Precision and Recall.

We have different combinations of epochs and batch sizes. An epoch describes the number of times the algorithm sees the entire data set. So, each time the algorithm has seen all samples in the dataset, an epoch has completed. Therefore, when we increase the number of epochs naturally the accuracy of the classifier increases. This was true in our observations also. Higher epochs led to higher accuracy. But increasing the number of epochs led to increase in time for training.

Model	Accuracy	Precision	Recall	f1 Measure
CNN	0.820250	0.803857	0.849826	0.826203
LSTM	0.871671	0.862508	0.883328	0.872794
CNN-LSTM	0.811500	0.811293	0.814520	0.812903
LSTM-CNN	0.806500	0.823992	0.782198	0.802551
ELMo	0.767250	0.727389	0.853781	0.785533
BERT	0.91925	0.9161736	0.923918	0.92002964

Table 5.1: Performance measures of models with data set size 20k

Our resources for conducting this research work was limited. Thus we couldnt increase the value of epochs too much. After trial and error, we fixed epoch as 3 for all the six models. In future work, running these models with higher epochs will lead to better results. Batch size is the number of training examples thats passed to the network during a single pass. We found that increasing batch size has little effect on the accuracy of classification. We have taken batch size as 16 in our training. We tried higher batch sizes, but a significant improvement in accuracy or other parameters were not found.

Table 5.1, shows the Accuracy, Precision, Recall and F1 score of the 6 models. For getting this result we used dataset of 20,000 reviews. In this we can see that BERT is the having the highest Accuracy, Precision, Recall and F1 score. Its followed by LSTM, CNN, CNN-LSTM, LSTM-CNN and ELMo. The following graphs shows a comparison of different models w.r.t different parameters.

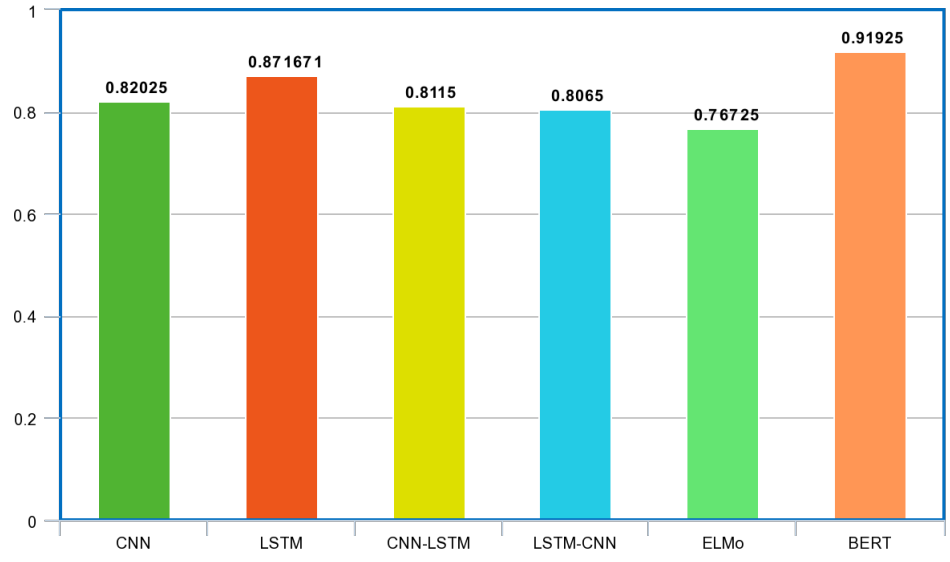


Figure 5.1: Accuracy comparison with data set size 20k

Table 2, shows the Accuracy, Precision, Recall and F1 score of the 6 models. For getting this result we used dataset of 100,000 reviews. In this we can see that BERT is the having the highest Accuracy,

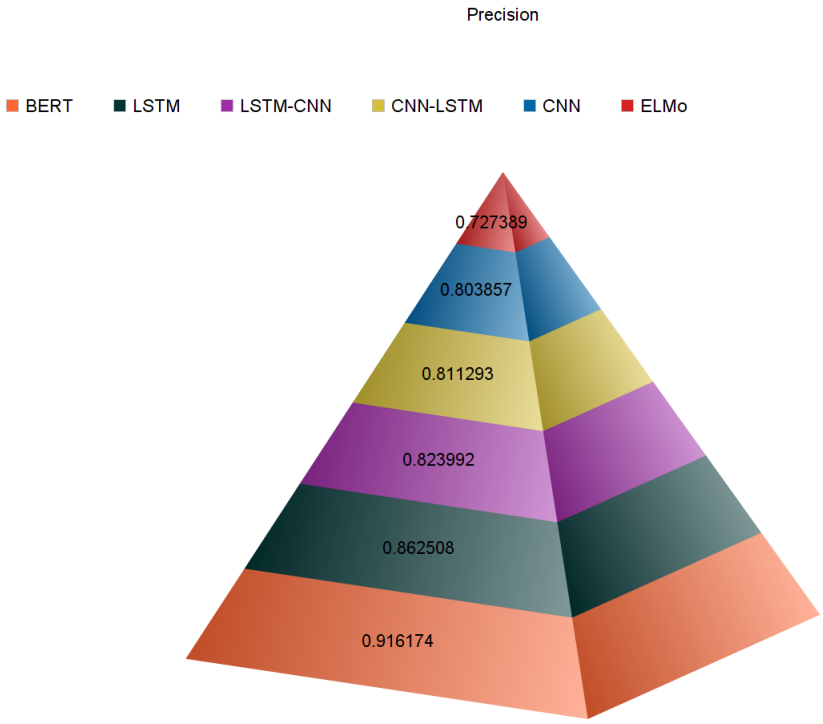


Figure 5.2: Precision comparison with data set size 20k

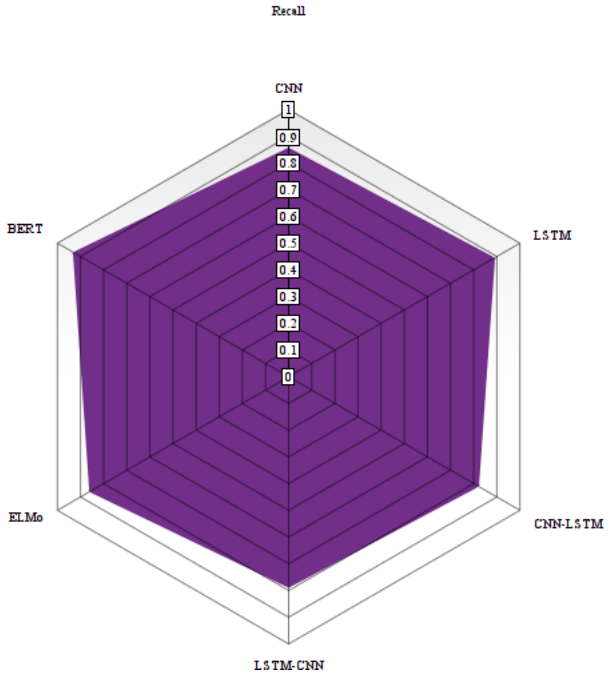


Figure 5.3: Recall comparison with data set size 20k

Precision, Recall and F1 score. Its followed by LSTM-CNN, CNN, CNN-LSTM, LSTM, and ELMo. The following graphs shows a comparison of different models w.r.t different parameters. As the dataset increased we can see

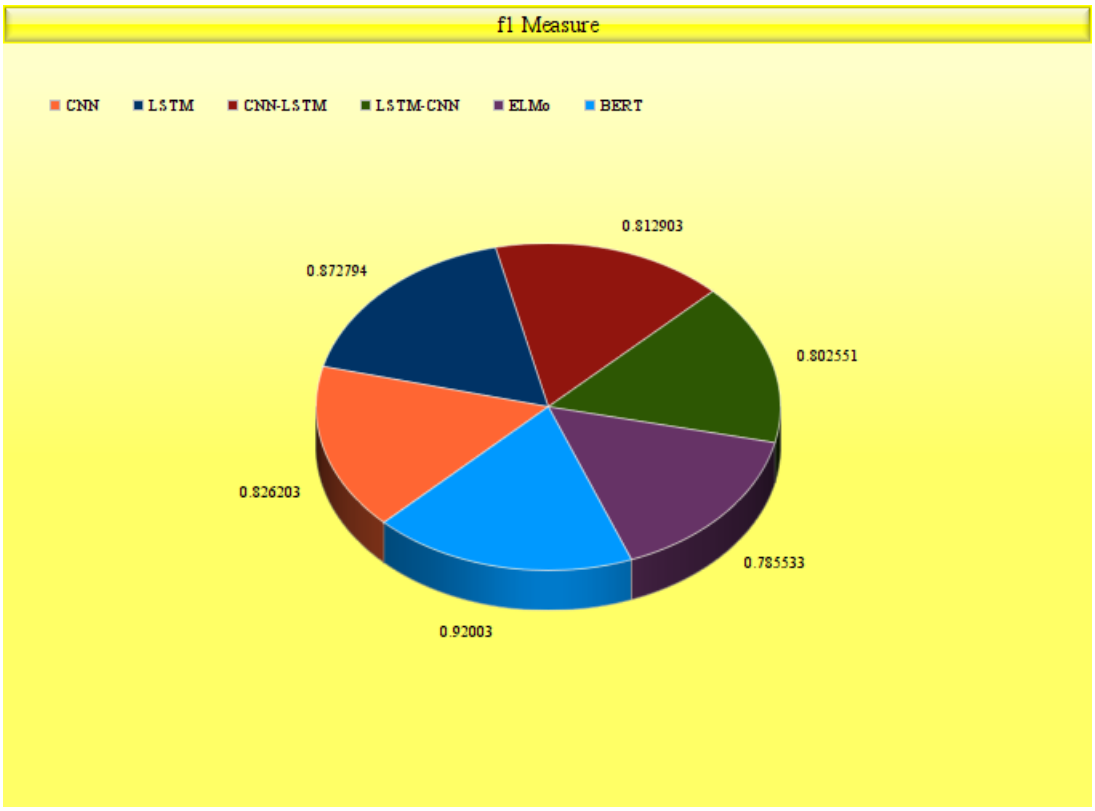


Figure 5.4: F1 score comparison with data set size 20k

Model	Accuracy	Precision	Recall	f1 Measure
CNN	0.837776	0.82697	0.85326	0.839909
LSTM	0.828585	0.803408	0.868924	0.834883
CNN-LSTM	0.829504	0.801817	0.874223	0.836456
LSTM-CNN	0.839154	0.851374	0.851374	0.835796
ELMo	0.801684	0.804241	0.792605	0.798381
BERT	0.94814795	0.94564784	0.95103425	0.9483334

Table 5.2: Performance measures Of models with data set size 100k

that the accuracy of models also increased.

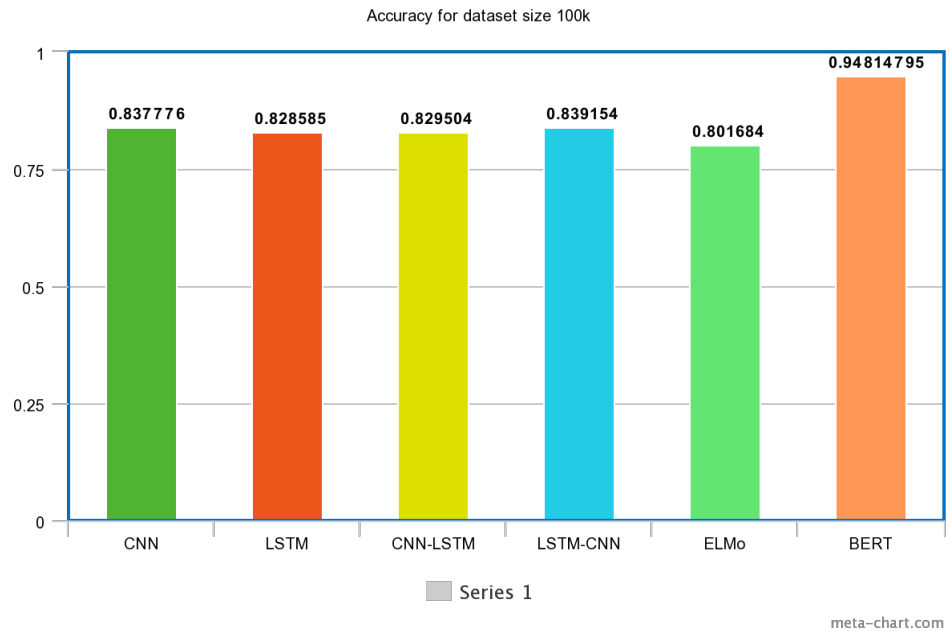


Figure 5.5: Accuracy comparison of models with data set size 100k

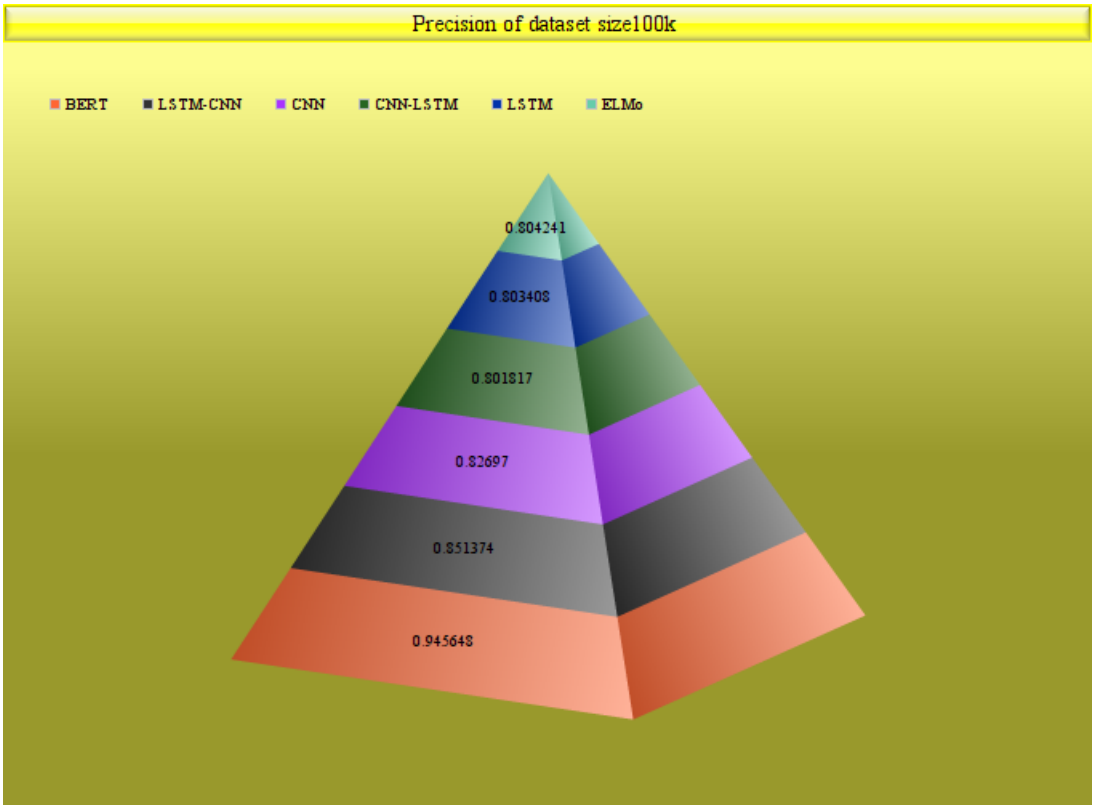


Figure 5.6: Precision comparison of models with data size 100k

Table 3, shows the Accuracy, Precision, Recall and F1 score of the 6 models. For getting this result we used dataset of 200,000 reviews. Here BERT is having the highest Accuracy, Precision, Recall and F1 score.

Figure 5.7: Recall comparison of models with data size 100k

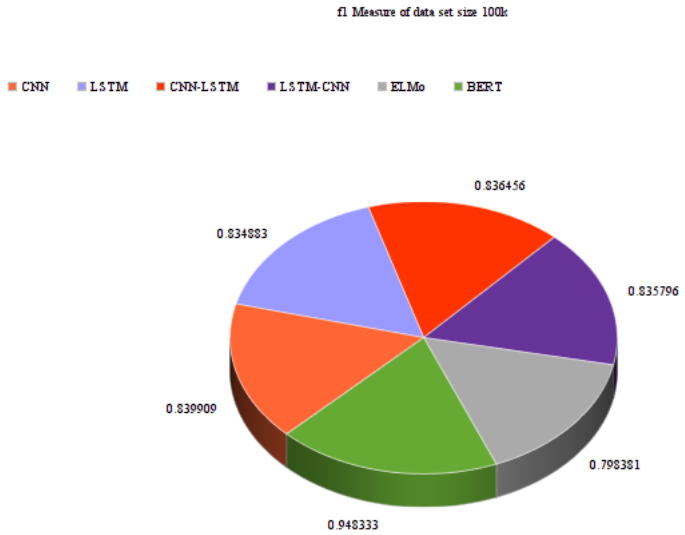


Figure 5.8: F1 score comparison of models with data size 100k

Model	Accuracy	Precision	Recall	f1 Measure
CNN	0.839629	0.870428	0.798466	0.832895
LSTM	0.85167	0.866471	0.844841	0.855519
CNN-LSTM	0.847022	0.830932	0.871751	0.850852
LSTM-CNN	0.846449	0.850834	0.840605	0.845689
ELMo	0.826781	0.850873	0.842645	0.808321
BERT	0.97769326	0.9775955	0.97779113	0.97769326

Table 5.3: Performance measures Of models with data set size 200k

Its followed by LSTM, CNN-LSTM, LSTM-CNN, CNN and ELMo. The following graphs shows a comparison of different models w.r.t different parameters. This is the maximum data set we used

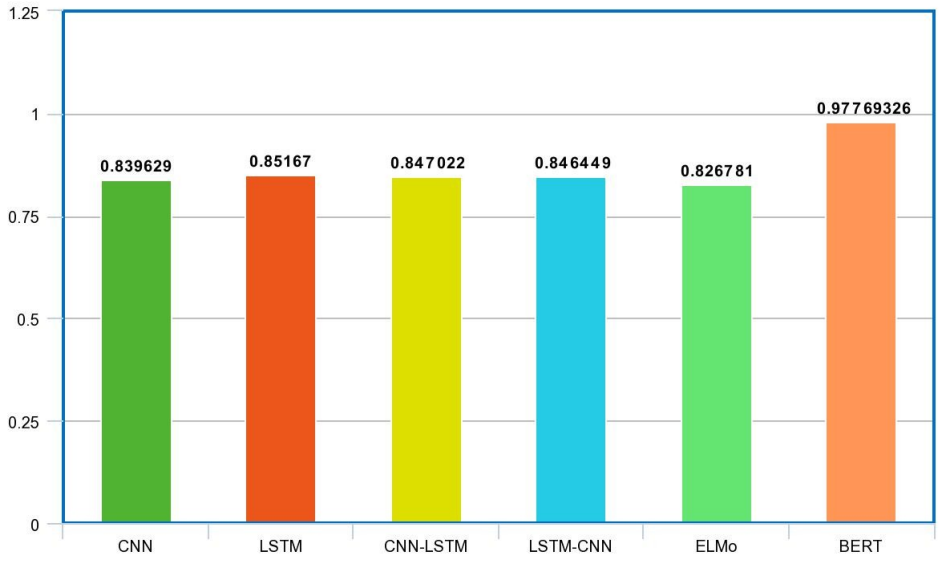


Figure 5.9: Accuracy comparison of models with data size 200k

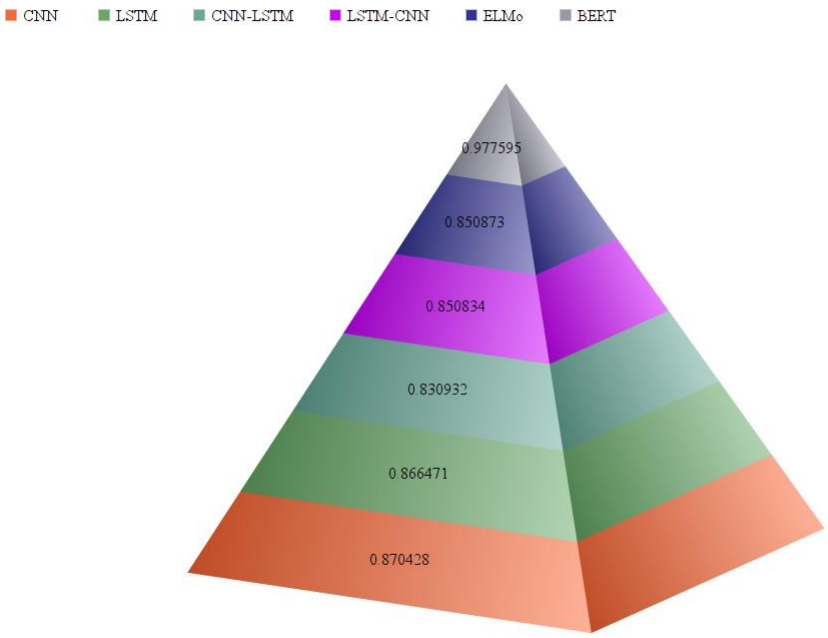


Figure 5.10: Precision comparison of models with data size 200k

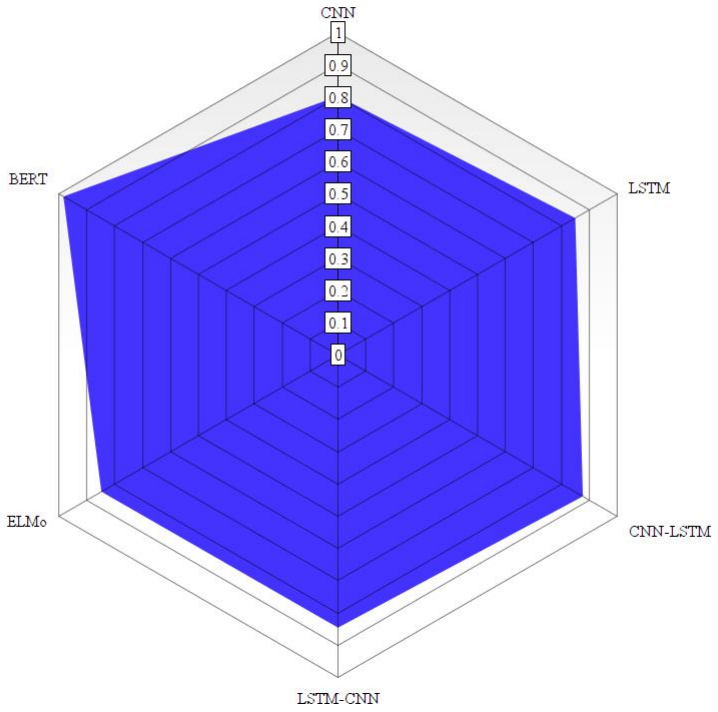


Figure 5.11: Recall comparison of models with data size 200k

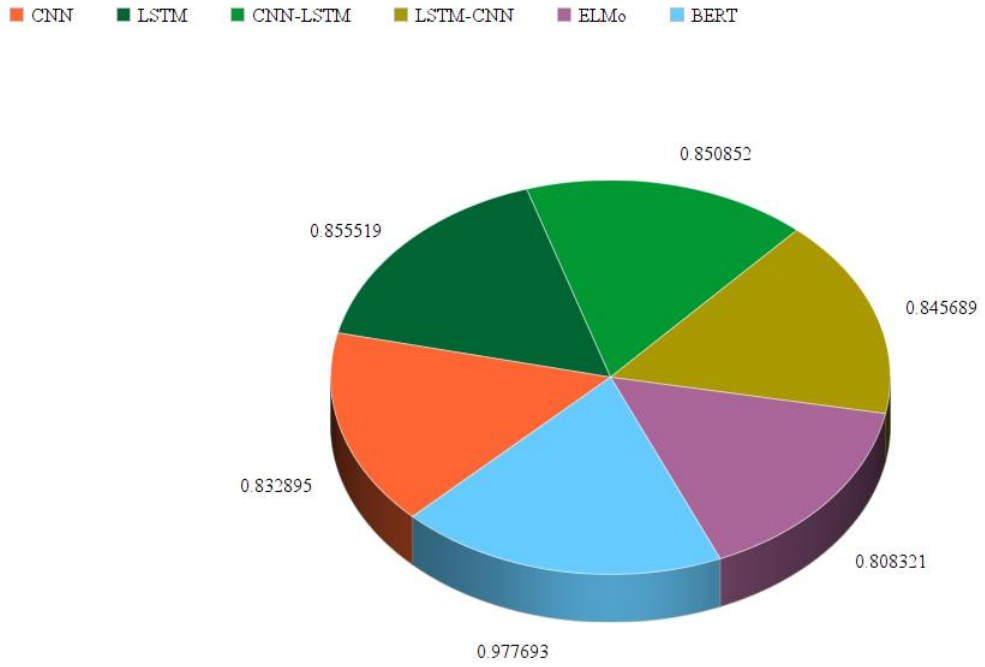


Figure 5.12: F1 score comparison of models with data size 200k

Chapter 6

Conclusion

Deep learning, which is a part of Machine Learning is gaining immense popularity and demand recently. Many researches are being carried out in this field all over the world. It's been utilised in fields such as computer vision, speech recognition, natural language processing and so on. It's predicted that the demand for Deep learning is going to rise in the future and deep learning market will be worth 18.16 billion USD by 2023.
In the world of social media, textual informations are created at a staggering rate and analysing them crucial. There comes the importance of Sentiment analysis.
Google's BERT which was released last October is widely regarded as a game changer in NLP. We were not able to find any proper research or study that compared it's performance with other Deep learning models. All these factors were the motivation for doing this project. In this project we have done Sentiment Analysis using six Deep learning models. Our aim was to find the best model among them.
We used Amazon food reviews as the data set. Our results shows that BERT is having the highest accuracy among all the models. This was an expected result since BERT is state of the art of what NLP can offer.

Future Work
For running deep learning models like BERT and ELMO, RAM and GPU requirements are high. We were unable to meet these high demands.

Thus we were forced to cut down the number of data sets used. If we could increase the number of data sets, then accuracy would have been better or maybe the results would have been slightly different. Running this in a powerful system which can handle large number of data sets can be considered as a future work for this project. Multi-Task Deep Neural Network (MT-DNN) is a NLP model created by Microsoft. It incorporates BERT for performing NLP. Comparing it's performance with BERT will be interesting. Publishing a research paper based on the project we did is another opportunity for future work.

Bibliography

[1] D. E. Rumelhartt G. E. Hinton, J. L. McClelland. Distributed representations. Technical report, 1986.

[2] Tomas Mikolov Quoc Le. Distributed Representations of Sentences and Documents. 2014.

[3] Geoffrey E. Hinton Ronald J. Williams. David E. Rumelhart. Learning representations by back-propagating errors. 1986.

[4] Pascal Vincent Christian Jauvin Yoshua Bengio, Rjean Ducharme. A neural probabilistic language model. *International ICT Conference MIPRO,*, 2006.

[5] A Collobert Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. 2008.

[6] Pascal Vincent Christian Jauvin Yoshua Bengio, Rjean Ducharme. A neural probabilistic language model. 2006.

[7] Manning and Schutze. Foundations of statistical natural language processing. 1999.

[8] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. 2008.

[9] Bing Liu. Sentiment analysis and opinion mining. 2012.

[10] Ting Liu Duyu Tang, Bing Qin. Document modeling with gated recurrent neural network for sentiment classification. 2004.

[11] Bo Pang and Lillian Lee. Thumbs up? sentiment classification using machine learning techniques. 2002.

[12] Paltoglou and 2010 Thelwall. A study of information retrieval weighting schemes for sentiment analysis. 2010.

[13] Navdeep Jaitly Noam Shazeer Samy Bengio, Oriol Vinyals. Scheduled sampling for sequence prediction with recurrent neural networks. 2015.

[14] Vasileios Hatzivassiloglou and Kathleen R. McKeown. Predicting the semantic orientation of adjectives. 1997.

[15] Michael L. Littman Peter D. Turney. Unsupervised learning of semantic orientation from a hundred-billion-word corpus. 2002.

[16] Corpora Marti A. Hearst. Automatic acquisition of hyponyms large text , 1992. 1992.

[17] Huettner and Subasic. Fuzzy typing for document management. 2000.

[18] Das and Chen. Yahoo! for amazon: Sentiment parsing from small talk on the web. 2001.

[19] Yuling Chen ; Zhi Zhang. Research on text sentiment analysis based on cnns and svm. 2018.

[20] Tong Zhang Rie Johnson. Effective use of word order for text categorization with convolutional neural networks. 2014.

[21] Andrew Senior Has im Sak Tara N. Sainath, Oriol Vinyals. Convolutional, long short-term memory,fully connected deep neural networks. 2009.

[22] Li Deng and John C.Platt. Ensemble deep learning for speech recognition. 2014.

[23] Mark Neumann et al Matthew E. Petersy. Deep contextualized word representations. March 2018.

[24] Kai Chen Greg S Corrado Tomas Mikolov, Ilya Sutskever and Jeff Dean. Distributed representations of words and phrases and their compositionality. 2013.

[25] Richard Socher Jeffrey Pennington and Christopher D. Manning. Glove: Global vectors forword representation. 2014.

[26] Jacob Goldberger Oren Melamud and Ido Dagan. Context2vec: Learning generic context embedding with bidirectional lstm. 2016.

[27] Sepp Hochreiter and Jurgen Schmidhuber. Long short term memory. 1997.

[28] Caiming Xiong Bryan McCann, James Bradbury and Richard Socher. Learned in translation: Contextualized word vectors. 2017.

[29] Mike Lewis Kenton Lee, Luheng He and Luke S. Zettlemoyer. End-to-end neural coreference resolution. 2017.

[30] Mike Schuster Qi Ge Thorsten Brants Phillipp Koehn Ciprian Chelba, Tomas Mikolov and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. 2014.

[31] Chandra Bhagavatula Matthew E. Peters, Waleed Ammar and Russell Power. Semi-supervised sequence tagging with bidirectional language models. 2017.

[32] Caiming Xiong Bryan McCann, James Bradbury and Richard Socher. Learned in translation: Contextualized word vectors. 2017.

[33] Jacob Devlin Ming-Wei Chang Kenton Lee and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018.

[34] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. in advances in neural information processing systems. 2015.

[35] Tim Salimans Alec Radford, Karthik Narasimhan and Ilya Sutskever. Improving language understanding with unsupervised learning. technical report, openai. 2018.

[36] Mohit Iyyer MattGardner Christopher Clark Kenton Lee Matthew Peters, Mark Neumann and LukeZettlemoyer. Deep contextualized word representations. 2018.

[37] Chandra Bhagavat-ula Matthew Peters, Waleed Ammar and Russell Power. Semi-supervised se-quence tagging with bidirectional language models. 2017.

[38] Jacob Devlin Ming-Wei Chang Kenton Lee and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018.

[39] Niki Parmar JakobUszkoreit Llion Jones Aidan N Gomez LukaszKaiser Ashish Vaswani, Noam Shazeer and Illia Polosukhin. Attention is all you need. 2017.

[40] Wojciech Zaremba Rafal Jozefowicz and Ilya Sutskever. An empirical exploration of recurrent network architectures. 2015.

[41] Chandra Bhagavatula Matthew E. Peters, Waleed Ammar and Russell Power. Semi-supervised sequence tagging with bidirectional language models. 2017.

[42] Klaus Greff Rupesh Kumar Srivastava and Jurgen Schmidhuber. Training very deep networks. 2015.

[43] Caglar Gulcehre Fethi Bougares Holger Schwenk Kyunghyun Cho, Bart van Merrienboer and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. 2014.

[44] Kyunghyun Cho Dzmitry Bahdanau and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. 2014.

[45] Alex Graves. Generating sequences with recurrent neural networks. 2013.