**Final Internship Report**

**On**

**Sentiment Analysis:**

**Using Transfer Learning for Amazon Reviews**

**At**

**XEBIA**

****

**Submitted On**

**28/ 07/ 2022**

**By**

**The DSML Team**

**Under the Supervision of**

**Mr. Harshit Dawar**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**DECLARATION**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

I declare that this written submission represents my ideas in my own words and where others'

ideas or words have been included, I have adequately cited and referenced the original

sources. I also declare that I have adhered to all principles of academic honesty and integrity

and have not misrepresented or fabricated or falsified any idea/data/fact/source in my

submission. I understand that any violation of the above will be cause for disciplinary action

by the Institute and can also evoke penal action from the sources which have thus not been

properly cited or from whom proper permission has not been taken when needed.

**Name of the student(s)-**

Abhishta Singh

Aditi Sharma

Aditi Singhal

Dev Watts

Nidhi Jha

Ravi Yadav

Richa Aggarwal

S. Mahima Manuela

Siddhartha Das Gupta

Tarun Tahiliani

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ACKNOWLEDGEMENT**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to our highly respected and esteemed guide, our industry mentor **Mr. Harshit Dawar** for his valuable guidance, encouragement and help for completing this project.Their useful suggestions for this whole work and cooperative behavior are sincerely acknowledged.

We would like to express my sincere thanks to our project mentor **Mr. Harikrishnan M** , for helping us throughout the project. I would also like to thank **Ms. Sanjana Mudgal** for her whole hearted support.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ABSTRACT**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

This project is a comparative study of **Sentiment Analysis** models, between the models of basic machine learning, deep learning and a study on different **transfer learning models**.These models are implemented on the dataset taken from Amazon Reviews in order to predict the sentiment of the rivews.

The objective of this project is to learn how to train machine learning models, build them, understand their differences, and also deploy sentiment analysis using the Python language.

The structure of this report is such that it starts with the study of basic machine learning models, their limitations, then a further study into deep learning models, some of their advantages and their limitations, then at the end to study transfer learning models and their advantages. In this project, the implementation of transfer learning models are done by using **Keras** for some models and **transformers** in others and to understand their differences.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**TABLE OF CONTENTS**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CHAPTER NO. TITLE PAGE NO**

DECLARATION

ACKNOWLEDGEMENT

ABSTRACT

Chapter 1:

INTRODUCTION

Chapter 2:

LITERATURE REVIEW

Chapter 3:

OBJECTIVE AND METHODOLOGY

Chapter 4:

BASIC MACHINE LEARNING MODELS

4.1 Multinomial Naive Bayes

4.2 Random Forest

4.3 K Nearest Neighbors

4.4 Logistic Regression

4.5 Support Vector Machines (SVM)

Chapter 5:

DEEP LEARNING AND TRANSFER LEARNING MODELS

5.1 RNN

5.2 GRU

5.3 LSTM

5.4 BI-LSTM

5.5 Multi Layer Perceptron

5.6 BERT with Neural Network

5.7 DistilBERT

5.7.1 DistilBERT

5.7.2 Libraries used

5.8 ALBERT

5.9 Electra

5.10 RoBERTa

Chapter 6:

COMPARATIVE STUDY

Chapter 7:

CONCLUSIONS AND RECOMMENDATION FOR FUTURE STUDY

REFERENCES

**CHAPTER 1**

**INTRODUCTION**

**1.1 Natural Language Processing (NLP)**

The capability of machines to understand and comprehend humanly understandable languages and the ability to assist humans by performing the required tasks such as translation, finding out the next sentence, predicting the missing word, suggesting phrases to complete the unfinished sentences (such as Google search), is called Natural Language Processing.

Since computers cannot understand raw text that can otherwise be understood by humans, the raw data first undergoes a crucial milestone known as **data preprocessing**. In this the data goes through a set of steps to be processed into a format such that it can be ‘digestible’ to the respective machine learning model that we wish to use. These steps are as follows-

1. **Tokenization**- In this step, the entire sentence gets converted into units called tokens, where each token can be a word, a punctuation mark, or a white space.
2. **Stop words**- In this step, the words of a sentence which are of less value are removed, such as the, are, am, it etc.
3. **Lemmatization**- In this step, the words get truncated such that only the root word remains.

**1.2 Sentiment Analysis**

One of the tasks of Natural Language Processing is **Sentiment Analysis**. These days the value of data is extremely high. Every company is hunting data from customers in order to improve their stand in the corporate world. Hence, understanding the sentiments of one’s customers is crucial. But, in today’s world, a well to do company might receive thousands, or even lakhs of reviews every day. Hence it is not feasible to manually read each and every review. In such a scenario, we have to turn to computer machines which can perform computation in seconds or even milliseconds. Hence, the idea behind Sentiment Analysis is to feed a bunch of reviews into a machine learning model (according to our choice) and to find out the probability of the predicted model for all the given sentiments. Thus, the sentiment holding the highest probability is considered as the sentiment of that particular review.

In this project, we train the following models-

1. Basic machine learning models
2. Basic deep learning models
3. Basic transfer learning models

**CHAPTER 2**

**LITERATURE REVIEW**

We have referred to the blog “**The Illustrated Transformer**” by Jay Alammar for understanding the working of a transformer.

The Transformer was proposed in the paper [Attention is All You Need](https://arxiv.org/abs/1706.03762). A TensorFlow implementation of it is available.. Harvard’s NLP group created a [guide annotating the paper with PyTorch implementation](http://nlp.seas.harvard.edu/2018/04/03/attention.html). In this post, we will attempt to oversimplify things a bit and introduce the concepts one by one to hopefully make it easier to understand to people without in-depth knowledge of the subject matter.

During training, an untrained model would go through the exact same forward pass. But since we are training it on a labeled training dataset, we can compare its output with the actual correct output.

**CHAPTER 3**

**OBJECTIVES AND METHODOLOGY**

**3.1 Objectives**

The aim of this project is to study the different types of machine learning models and to prepare a comparative study of all the models. It is also to introduce transfer learning models, and the advantage of using transformers for processing texts in NLP. It is also to deploy the best model for sentiment of reviews by training on a bunch of Amazon Review dataset.

**3.2 Methodology**

We first start off by carrying out data pre processing of the given dataset and training the resultant corpus on some basic machine learning models in order to understand the fundamentals of machine learning.

Then we use some deep learning models in order to train models by learning from mistakes by introducing neurons and training layers. But even in this we have one major disadvantage which is the lack of parallel computation.

In order to overcome this, we use something called transformers which ensures parallel processing, hence introducing **Attention** and also **Self Attention** which enables a training model to understand the underlying meaning of a human understandable language, and thus to give suitable response.

We can simply use transformers by using models that are specific, either as encoders or decoders. For example, BERT or **B**idirectional **E**ncoder **R**epresentations from **T**ransformers is used as the encoder in a transformer. On the other hand, GPT-3 or third generation **G**enerative **P**re-trained **T**ransformer is used as the decoder part of a transformer.

We can use these encoders or decoders in our model but building them from scratch is not feasible since they have been trained on millions of data, across various resources such as wikipedia, newsletters etc.

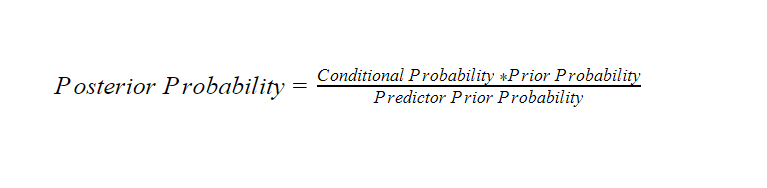
Hence, we can use them ready-made via Transfer Learning which ensures us to use pre-trained models in our use-case by just replacing the last layer of the pre-trained model in order to perform the task which we wish to carry out.

**CHAPTER 4**

**BASIC MACHINE LEARNING MODELS**

**4.1 Multinomial Naive Bayes**

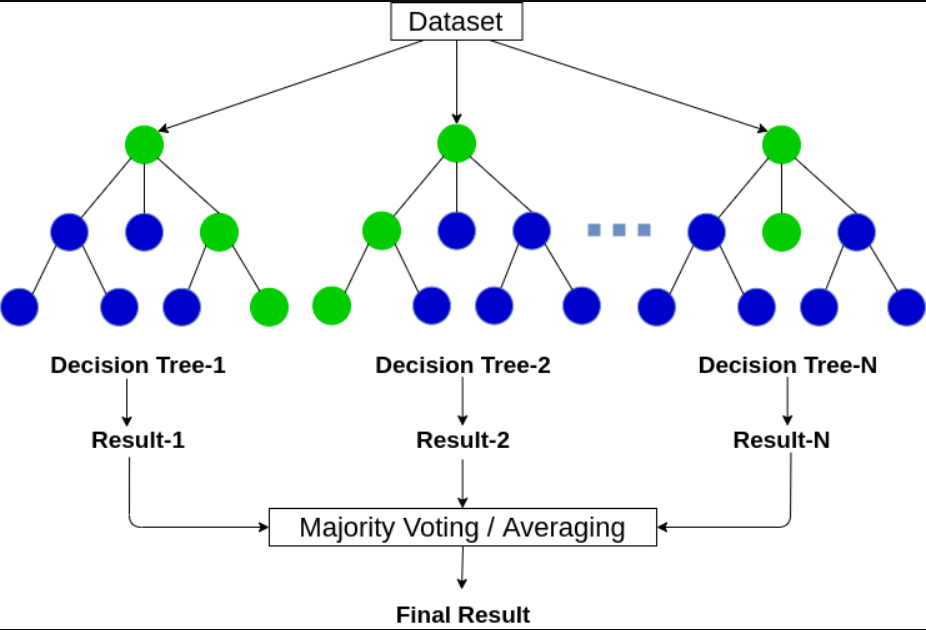
The Multinomial Naive Bayes classifier is one of the simpleset classifier for classifying texts and it works on the concept of posterior probability-



In this the posterior probability for a review being positive and negative is calculated separately. For example, for calculating the posterior probability for the sentiment of a review to be positive-

1. First the **conditional probability** is calculated as the number of positive sentiments in the entire dataset divided by the total number of reviews.
2. The prior probability is calculated separately for every word in the review. It is calculated as what is the probability for a review to be positive if that particular word is present in the review. After calculating its probability separately for each word in the entire bag of words, the summation is taken at the end
3. The predictor prior probability can be ignored as it is a summation of all the reviews in the dataset, which is a constant.

**4.2 Random Forest**

Random Forest is an ensemble model meaning, it is formed by aggregation of multiple models. In the case of a random forest , it is formed by combining multiple decision trees. This model can be used for both classification and regression problems. The disadvantage of using a single decision tree model is that it produces less accurate results due to high bias and low variance. But an aggregate of these decision trees gives better results due to the fact that uncorrelated models perform better in groups than they do alone. Each of the trees provide a classification of the input data. The classification which is in majority, is displayed as the output. 

In our project the random forest classification model is applied to both TF-IDF and BoW vectorized data. Number of trees used is 400 which is mentioned with the help of the parameter ‘n\_estimators’ and the criterion used is ‘gini’ impurity. Gini impurity is used to measure the quality of the split but the main difference between gini and entropy criteria is that the impurity in gini will peak at 0.5 whereas for entropy it will peak at 1. Moreover, the efficiency of gini is better than the entropy criterion. After evaluating the model with both the vectorization techniques, it is concluded that accuracy with Bow is 88% and with TF-IDF is 84%. Since it is known that accuracy shouldn’t be the only metric to evaluate any model, we also calculated precision, recall and f1 score of both the models. Even then the better performing model remains the same, so we can say that the model with BoW is working better than TFIDF.

**4.3 K Nearest Neighbors**

K-Nearest neighbors (KNN) calculation is a sort of supervised machine learning calculation which can be utilized for both grouping as well as relapse prescient issues. Hence, it is mostly utilized for grouping prescient issues in industry.

K-NN calculation expects the closeness between the new case/information and accessible cases and put the new case into the classification that is generally like the accessible classifications.

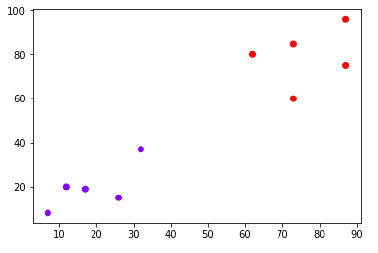
K-NN calculation stores every one of the accessible information and characterizes another information point in view of the comparability. This implies when new information shows up then it tends to be handily grouped into a well-suited class by utilizing K-NN calculation.

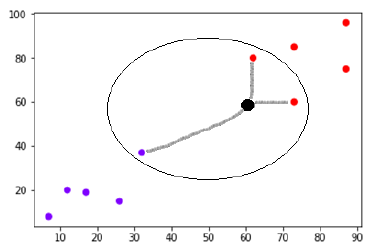
K-NN calculation can be utilized for Regression as well concerning Classification yet for the most part it is utilized for the Classification issues.

K-NN is a non-parametric calculation, and that implies it makes no presumption on hidden information.

It is likewise called a languid student calculation since it doesn't gain from the preparation set quickly rather it stores the dataset and at the hour of grouping, it plays out an activity on the dataset.

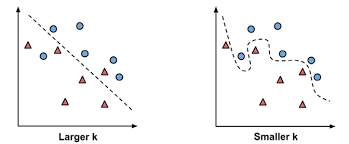
KNN calculation at the preparation stage simply stores the dataset and when it gets new information, then it groups that information into a classification that is much like the new information.



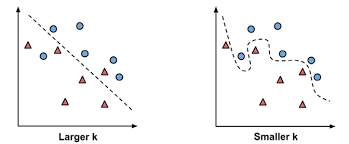


**The effect of choosing a smaller or larger K worth on the model**

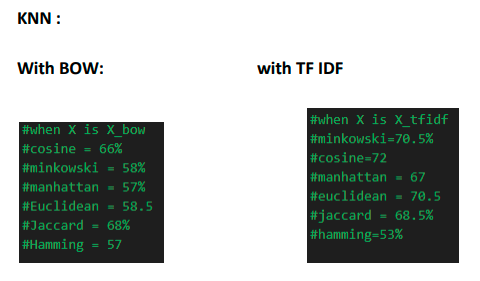
**Large K Value:** The instance of underfitting happens when the worth of k is expanded. For this situation, the model would not be able to accurately learn on the preparation information.



**Smaller K Value:** The state of overfitting happens when the worth of k is more modest. The model will catch all of the preparation information, including clamor. The model will perform inadequately for the test information in this situation.

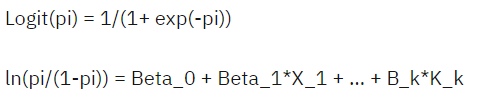


**Output:**



**4.4 Logistic Regression**

This type of statistical model (also known as *logit model*) is often used for classification and predictive analytics. Logistic regression estimates the probability of an event occurring, such as voted or didn’t vote, based on a given dataset of independent variables. Since the outcome is a probability, the dependent variable is bounded between 0 and 1. In logistic regression, a logit transformation is applied on the odds—that is, the probability of success divided by the probability of failure. This is also commonly known as the log odds, or the natural logarithm of odds, and this logistic function is represented by the following formulas:

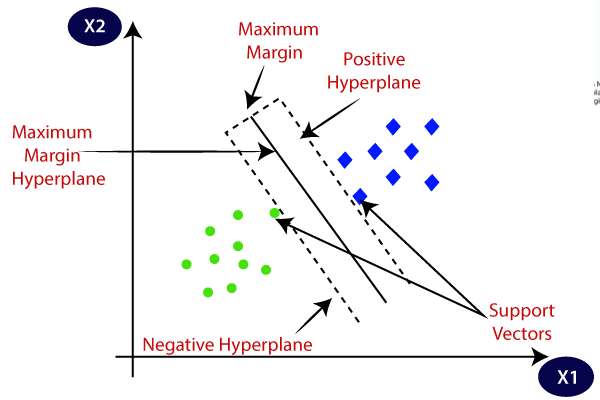
****

In this logistic regression equation, logit(pi) is the dependent or response variable and x is the independent variable. There are three types of logistic regression models, which are defined based on categorical response:

1. Binary logistic regression: In this approach, the response or dependent variable is dichotomous in nature—i.e. it has only two possible outcomes (e.g. 0 or 1).
2. Multinomial logistic regression: In this type of logistic regression model, the dependent variable has three or more possible outcomes; however, these values have no specified order.
3. Ordinal logistic regression: This type of logistic regression model is leveraged when the response variable has three or more possible outcomes, but in this case, these values do have a defined order.

Within [machine learning](https://www.ibm.com/in-en/cloud/learn/machine-learning), logistic regression belongs to the family of [supervised machine learning](https://www.ibm.com/in-en/cloud/learn/supervised-learning) models. It is also considered a discriminative model, which means that it attempts to distinguish between classes (or categories).It maximizes the log likelihood function to determine the beta coefficients of the model. This changes slightly under the context of machine learning. Within machine learning, the negative log likelihood is used as the loss function, using the process of [gradient descent](https://www.ibm.com/in-en/cloud/learn/gradient-descent) to find the global maximum. [Scikit-learn](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) (link resides outside IBM) provides valuable documentation to learn more about the logistic regression machine learning model.

**4.5 Support Vector Machine(SVM)**

Support Vector Machine (SVM) is a supervised machine learning algorithm that is used for both classification and regression. Although we also refer to regression concerns, categorization is the most appropriate term. Finding a hyperplane in an N-dimensional space that clearly classifies the data points is the goal of the SVM method. The number of features determines the hyperplane's size. The hyperplane is essentially a line if there are just two input features. The hyperplane turns into a 2-D plane if there are three input features. Imagining something with more than three features gets challenging. Advantages of this model being able to be effective in high dimensional cases, being memory efficient as it uses a subset of training points in the decision function called support vectors and it also has different kernel functions can be specified for the decision functions and it is possible to specify custom kernels.

The SVM algorithm's objective is to establish the best line

or decision boundary that can divide n-dimensional space

into classes, allowing us to quickly classify fresh data points

in the future. A hyperplane is the name given to this optimal

decision boundary.

SVM selects the extreme vectors and points that aid in the creation of the hyperplane. Support vectors, which are used to represent these extreme instances, form the basis for the SVM method. Consider the diagram, where a decision boundary or hyperplane is used to categorize two distinct categories. In our model we implemented Support Vector Machine on TF-IDF vectorized data and got the accuracy as 82%.

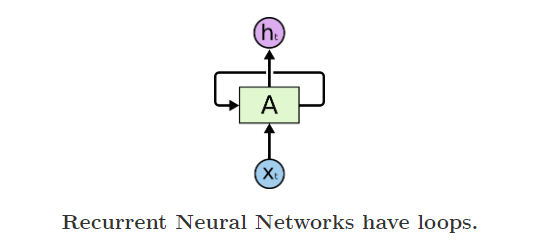
**CHAPTER 5**

**DEEP LEARNING AND TRANSFER LEARNING MODELS**

**5.1 Recurrent Neural Network**

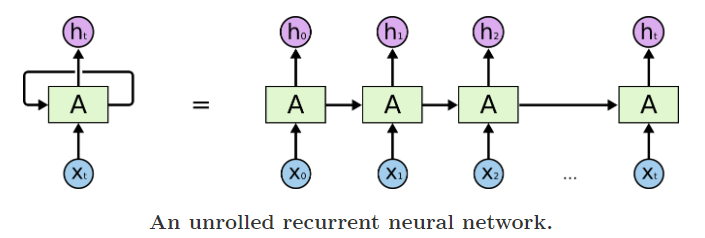
**5.1.1 Overview**

Traditional Neural Networks are unable to learn from their previous outputs in order to go further. Human brain is capable of learning from the previous events and applying it as an “input” for the current event. Hence, in order to address the issue raised by the Traditional Neural Networks , Recurrent Neural Networks i.e. RNN fulfilled that. They are networks with loops in them, allowing information to persist.



In the diagram above, neural network fragment A sees some input xt and outputs the value of ht. Loops allow you to pass information from one network level to another. These loops give the recurrent neural network a mysterious look. However, if you think about it a little, you can see that it is not very different from a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to its successor.

**5.1.2 Architecture**



Suppose there is a deeper network with one input layer, three hidden layers and one output layer. Then like other neural networks, each hidden layer will have its own set of weights and biases, let’s say, for hidden layer 1 the weights and biases are (w1, b1), (w2, b2) for second hidden layer and (w3, b3) for third hidden layer. This means that each of these layers are independent of each other, i.e. they do not memorize the previous outputs.

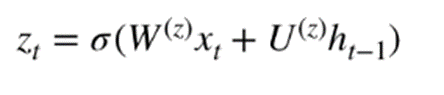
RNN converts the independent activations into dependent activations by providing the same weights and biases to all the layers, thus reducing the complexity of increasing parameters and memorizing each previous output by giving each output as input to the next hidden layer.

Hence these three layers can be joined together such that the weights and bias of all the hidden layers is the same, into a single recurrent layer.

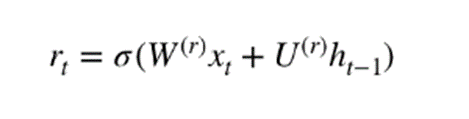
* Formula for calculating current state: ht = f(ht-1, xt), where ht, ht-1, xt are current state, previous state and input respectively.
* Formula for applying Activation function(tanh): ht = tanh(Whhht-1+wxhxt), where whh, wxh are weight at recurrent neuron, weight at input neuron respectively.
* Formula for calculating output: yt = whyht, where yt is output and why is weight at output layer.

**5.2 Gated Recurrent Unit**

Similar to LSTM, GRU also aims at reducing the vanishing gradient problem. The main difference between GRU and LSTM is in the number of gates used by the models. LSTM uses 3 gates whereas GRU uses 2 gates namely reset and update gates. These gates decide which information is to be forwarded to the output.

1. Update Gate(zt)- This gate is used to determine which of the previous information is needed to be passed to the next cell state.

When the current information (xt)is inserted into the network it gets multiplied by its own weight (Wz). Similarly, the previous information(h(t-1)) is multiplied by its own weight (Uz). Both the products are added and a sigmoid activation function is applied to transform the result in the range of 0 and 1.

2. Reset Gate(rt)- This gate determines which of the previous information is not needed and can be discarded. The formula comes out to be the same as update gate 

After the required preprocessing of the data, that is, data cleaning, removing stop words, lemmatization and creating a corpus, the cleaned data is divided for training and testing in the ratio of 80:20. Then the data undergoes text tokenization which is inbuilt in the ‘keras.preprocessing ‘ library. For that, it takes the maximum number of words to keep based on the frequency of the same. We have taken all the words in the vocabulary. In the next step we fit the tokenization function into the train data. Then the tokenized text along with the training text is converted into a sequence of tokens. Both the sequences are padded to make the length of the tokens equal. For that, the maximum length of the review present in the dataset is calculated and is used as a parameter in the padding function.

For building the GRU model, an embedding layer is added having 128 neurons. Stack of 128 GRU models is then added to the model and lastly for the output, a dense layer is added. 50% of the neurons are dropped in the first layer to prevent overfitting of the model. Activation function of sigmoid is used since it is generally used for binary classification, giving an output between 0 and 1.

Lastly the model is compiled mentioning the loss function as ‘binary cross entropy’ and optimizer as ‘adam. Binary Crossentropy loss function is used for the same reason of the problem statement being a binary classification problem and for the optimizer which is used to increase the efficiency of the model, Adam which is proved to be better than rest by a comparative study by various researchers.

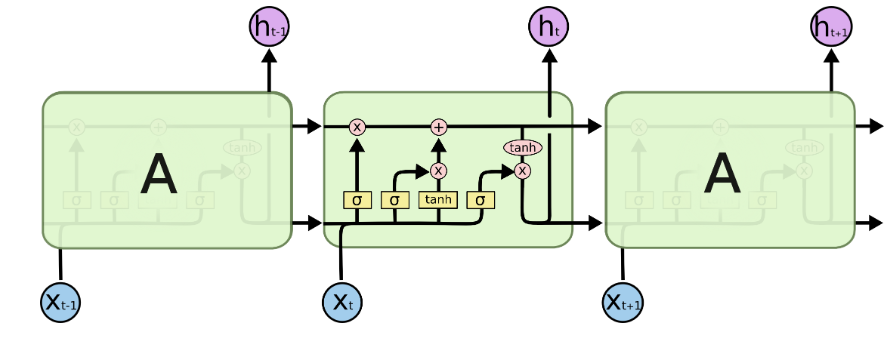
To train the model, epoch of 200 iterations is mentioned and batch size of 64 meaning the number of random data points to be taken into consideration while running an epoch is used. The train and test results are calculated using evaluate function which comes out to be 99% and 83% respectively

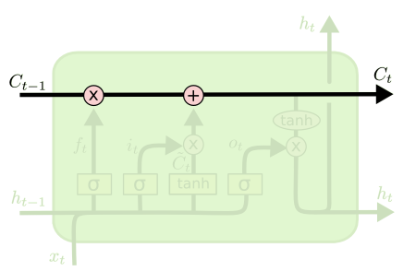
**5.3 Long Short Term Memory**

**5.3.1 Overview**

Long Short Term Memory networks – usually just called LSTMs – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by [Hochreiter & Schmidhuber (1997)](http://www.bioinf.jku.at/publications/older/2604.pdf), and were refined and popularized over time. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

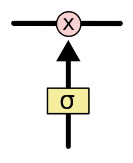
The repeating module in a standard RNN contains a single layer whereas in LSTM they have this chain-like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



**5.3.2 Architecture**

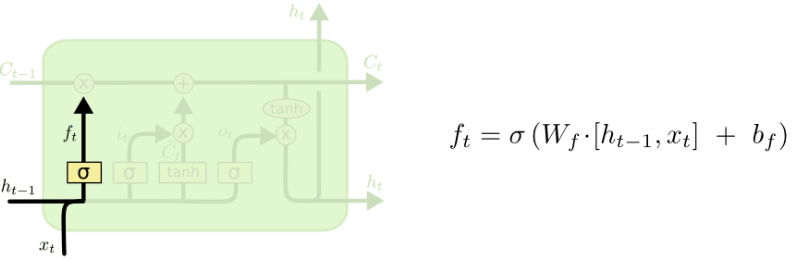
The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It’s very easy for information to just flow along it unchanged.

The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

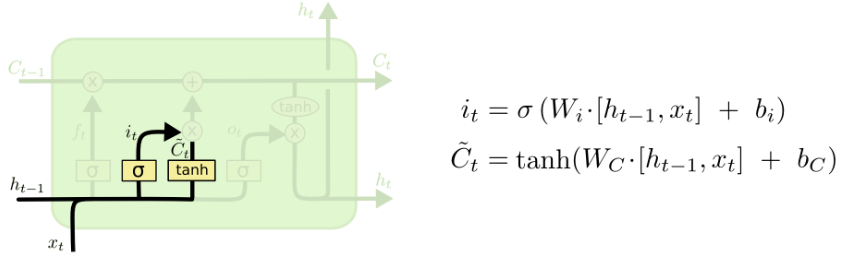
The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!” An LSTM has three of these gates, to protect and control the cell state.

5.3.3 Steps

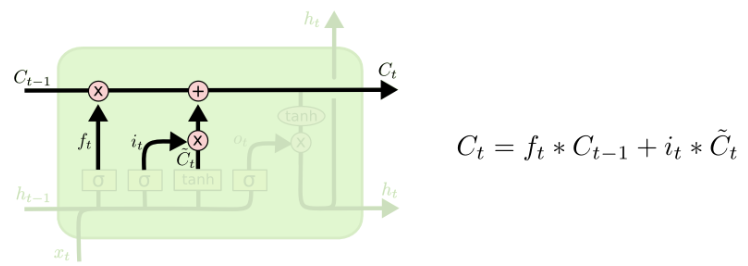
The first step in our LSTM is to decide what information we’re going to throw away from the cell state. This decision is made by a sigmoid layer called the forget gate layer.



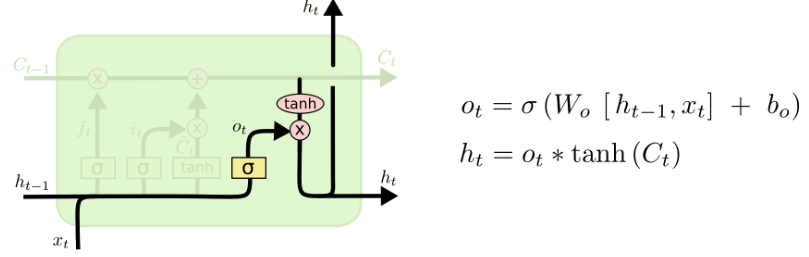
The next step is to decide what new information we’re going to store in the cell state. This has two parts: The sigmoid layer and a tan layer.



In the third step we have to update the cell state, the previous steps have decided what to do, we just have to do it.



The final step is the output layer, this output will be based on our cell state, but will be a filtered version.

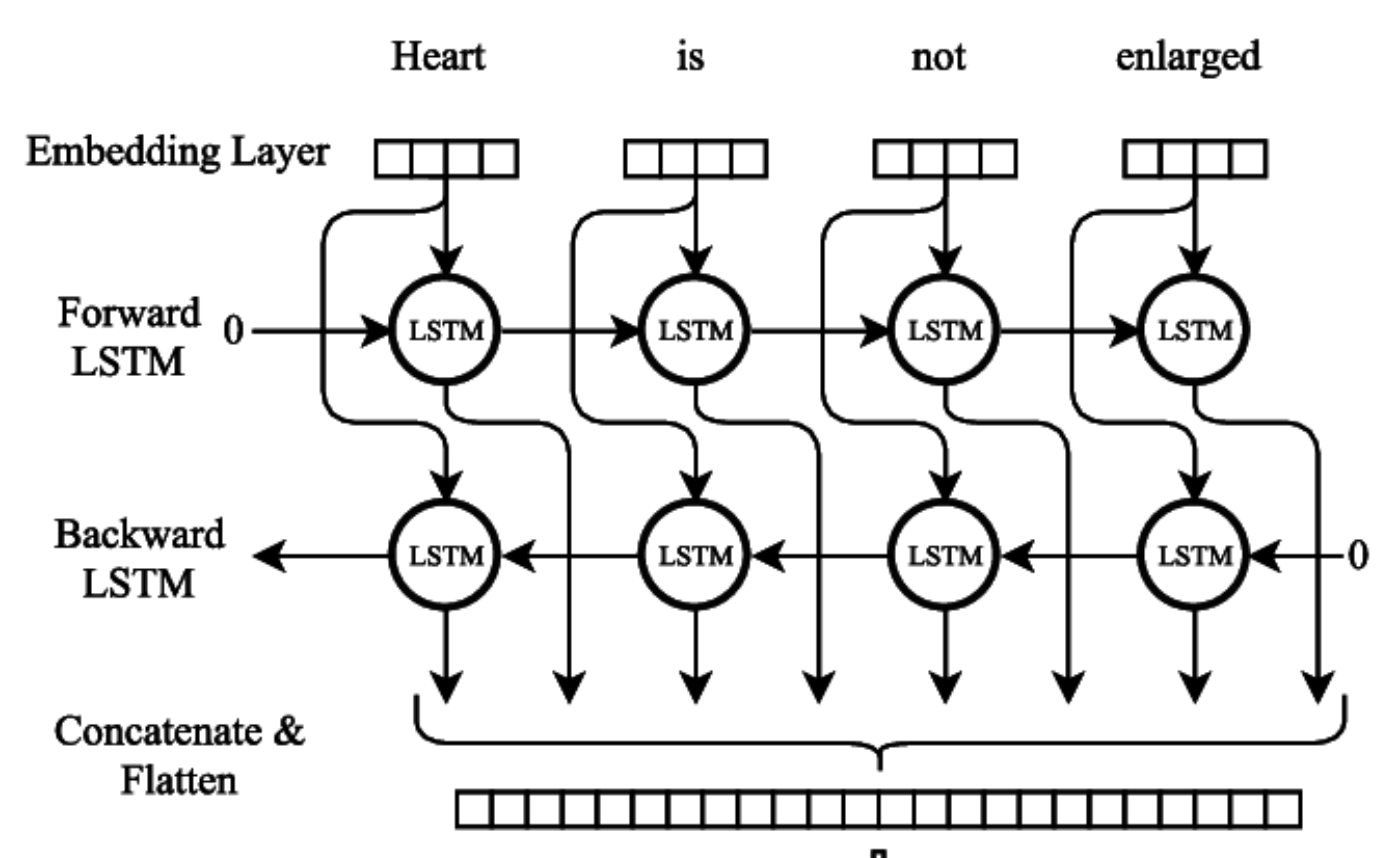


**5.4 Bidirectional - LSTM**

**5.4.1 Overview**

A Bidirectional LSTM, or biLSTM, is a sequence processing model that consists of two LSTMs: one taking the input in a forward direction, and the other in a backwards direction. BiLSTMs effectively increase the amount of information available to the network, improving the context available to the algorithm.To train the model, epoch of 20 iterations is mentioned and batch size of 32 meaning the number of random data points to be taken into consideration while running an epoch is used. The train and test results are calculated using evaluate function which comes out to be 97% and 76% respectively

**5.4.3 Architecture**



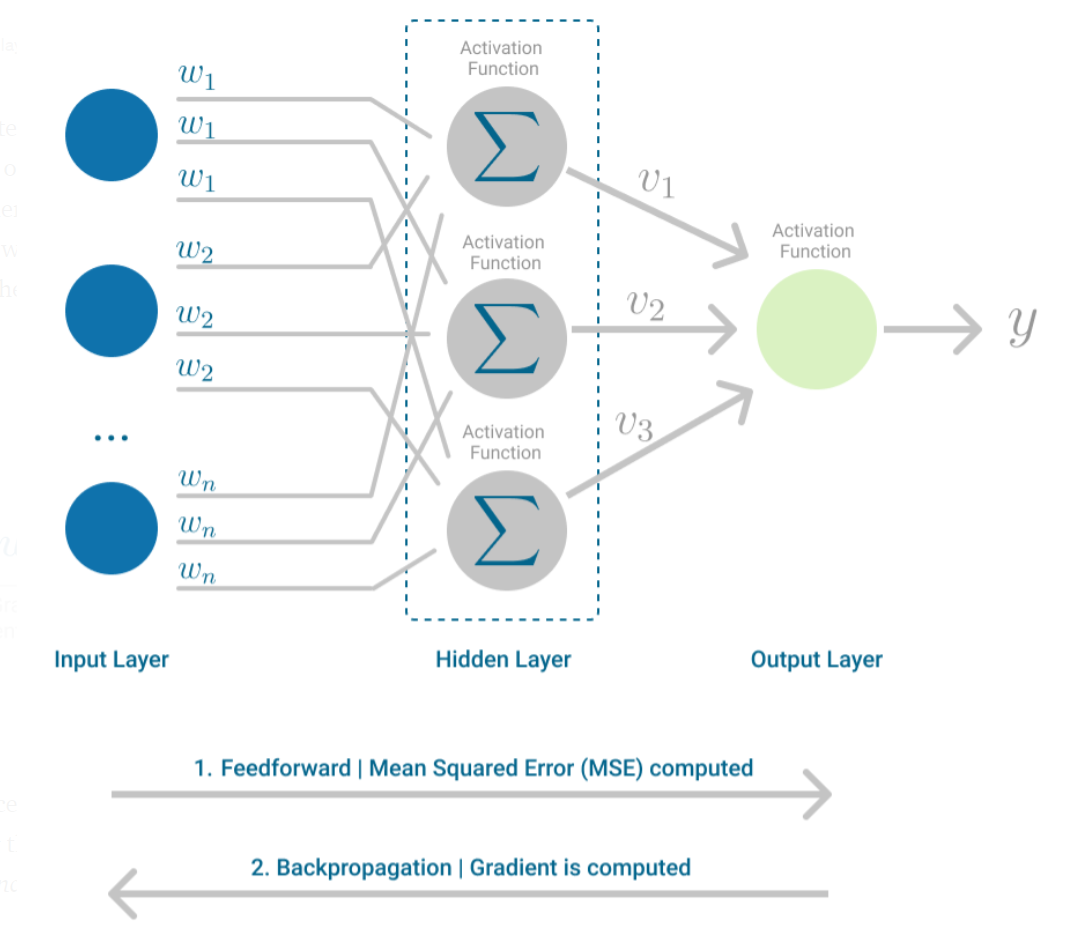
Using bidirectional will run your inputs in two ways, one from past to future and one from future to past and what differs this approach from unidirectional is that in the LSTM that runs backward you preserve information from the future and using the two hidden states combined you are able in any point in time to preserve information from both past and future.

**5.5 Multi Layer Perceptron (Deep Learning)**

A Multilayer Perceptron has input and output layers, and one or more hidden layers with many neurons stacked together. Each layer is *feeding* the next one with the result of their computation, their internal representation of the data. This goes all the way through the hidden layers to the output layer.

If the algorithm only computed the weighted sums in each neuron, propagated results to the output layer, and stopped there, it wouldn’t be able to *learn* the weights that minimize the cost function. If the algorithm only computed one iteration, there would be no actual learning.

This is where Backpropagation comes into play.



Backpropagation is the learning mechanism that allows the Multilayer Perceptron to iteratively adjust the weights in the network, with the goal of minimizing the cost function.

**Model Architecture**

* First layer is of word embedding layer which will give us the real valued vectors which are similar for similar words
* we can specify the length of each vector for word embedding
* second layer is of global average pooling layer which reducing the chances of overfitting by reducing the no. of parameters by storing the average of each value of ith index
* Third layer is dense layer which have 16 hidden units which uses relu as an activation function
* Final layer is our o/p layer which uses sigmoid as an activation function, the output layer gives us the floating point value which lies between 0-1

**Model Compilation**

* Then to compile this model I used adam optimizer as its one of the best optimizer for NN
* Binary cross entropy loss for loss function for model
* Accuracy, precision, recall , F1 Score as a metric to evaluate the model

**Model Results**

* Accuracy : 85.5%
* Precision : 83.8%
* Recall : 86.4%
* F1 Score : 85.12%

**5.6 BERT with Artificial Neural Network**

**5.6.1 Overview**

Bidirectional Encoder Representations from Transformers(BERT) is based transformer architecture, pre trained on large corpus of unlabelled text including the entire Wikipedia (that’s 2,500 million words) and Book Corpus (800 million words). As it is trained on a large text corpus, our model starts to pick up the deeper and intimate understandings of how the language works. BERT is deeply bidirectional” model that is it learns information from both the left and the right side of a token’s context during the training phase.

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.

BERT uses two training strategies. Masked LM(MLM) and Next sentence prediction(NSP). In 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

* Adding a classification layer on top of the encoder output.
* Multiplying the output vectors by the embedding matrix, transforming them into the vocabulary dimension.
* Calculating the probability of each word in the vocabulary with softmax.

Next sentence prediction:

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.

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.

**5.7 DistilBERT**

**5.7.1** **Overview**

The DistilBERT model is a refined model of the BERT base model, having been trained on the same dataset as its BERT counterpart. It is a compact version of BERT and it is also trained in less time than BERT, thus making it a more portable model than BERT.

This model is a compact version of BERT as its parameters are forty percent lesser than that of BERT. It also retains about ninety percent accuracy of its BERT counterpart.

The DistilBERT model is trained on two models in order to get embeddings and understand/ process human language-

1. Finding the masked words in a sentence
2. Predicting the next sentence after a given sentence (input) called NSP

This model acts as the encoder part of the transformer used for NLP, text classification. Hence, this model is mainly used to understand the semantic meaning of the natural language that it receives as input in order to facilitate the task that needs to be carried out.

On the other hand, on the decoder side, the last layer of the DistilBERT is removed and replaced with the layer according to the situation. By default this model is used for finding the masked language and for next sentence prediction but we can fine-tune our model according to our requirements. For example, since in our scenario we have to perform sentiment analysis, we can simply use the model AutoModelForSequenceClassification (from *hugging face*). This model acts as the last layer of the pretrained model which we have defined.

**5.7.2 Libraries used**

1. **Transformers**- in order to use DistilBERT (the pretrained model), including the AutoTokenizer for tokenizing the raw data
2. **Datasets**- for converting the raw data present in the form of dataframe into dataset in order to feed it into the *hugging face* model.
3. **Pandas**- for working with the dataframes
4. **Numpy**- for dealing with the output predicted value and accuracy
5. **Sklearn**- for splitting the train and test data separately

**5.8 ALBERT:**

Albert is a transformers model pre-trained on an enormous corpus of English information in a self-regulated style. This implies it was pre trained on the crude texts just, without any people naming them in any capacity (which is the reason it can utilize heaps of openly accessible information) with a programmed cycle to produce data sources and marks from those texts. All the more definitively, it was pretrained with two goals:

* **Masked language modeling (MLM):** taking a sentence, the model randomly covers 15% of the words in the info then, at that point, run the whole veiled sentence through the model and needs to foresee the veiled words. This is unique in relation to conventional recurrent neural networks (RNNs) that normally see the words in a steady progression, or from autoregressive models like GPT which inside veil the future tokens. It permits the model to gain proficiency with a bidirectional portrayal of the sentence.
* **Sentence Ordering Prediction (SOP):** ALBERT utilizes a pre training method which helps in finding the correct sequence of sentences in any unordered document.

Along these lines, the model learns an inward portrayal of the English language that can then be utilized to separate highlights helpful for downstream errands: in the event that you have a dataset of marked sentences for example, you can prepare a standard classifier utilizing the elements created by the ALBERT model as sources of info

ALBERT is specific in that it shares its layers across its Transformer. In this manner, all layers have similar loads. Utilizing rehashing layers brings about a little memory impression, nonetheless, the computational expense stays like a BERT-like engineering with a similar number of stowed away layers as it needs to emphasize through a similar number of (rehashing) layers.

This is the second form of the base model. Form 2 is not the same as rendition 1 because of various dropout rates, extra preparation information, and longer preparation. It has improved results:

* 12 repeating layers
* 128 embedding dimension
* 768 hidden dimension
* 12 attention heads
* 11M parameters

Known as ‘A lite version of BERT’, ALBERT was proposed as of late to improve the preparation and consequences of BERT engineering by utilizing boundary sharing and factorizing procedures. BERT model contains a large number of boundaries, BERT-based holds around 110 million boundaries which makes it hard to prepare likewise such a large number of boundaries influence the calculation. To defeat such difficulties ALBERT was presented as It has less boundaries contrasted with BERT. ALBERT utilizes two strategies:

**Cross-Layer Parameter Sharing**It's a procedure utilized for diminishing the quantity of boundaries in BERT. BERT contains N number of encoder layers, for instance, BERT-base has 12 encoder layers. During preparing, the boundaries are advanced across all encoder layers. With regards to cross-layer. Rather than learning boundaries across all encoder layers. The boundary of the first encoder layer is imparted to the wide range of various encoder layers. Just the first encoder layer is extended. Each encoder contains sub-layers. Multi Headed consideration and feedforward. We get familiar with the boundaries of encoder 1 and offer them across with other encoder layers.

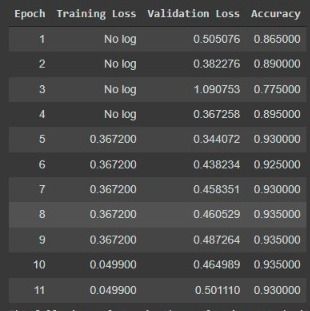
* All-shared – All boundaries of the encoder layer, including all the sub-layers (multi-head consideration layer and feed-forward layer) are shared across all encoder layers of the BERT model.
* Feed-forward – Just the feed-forward layer boundaries are imparted to the feed-forward sub-layers across the wide range of various encoder layers of the BERT model.
* Shared attention – Only the consideration head layer boundaries are imparted to the consideration head sub-layers across all encoder layers of the BERT model.

By default, ALBERT utilizes an all-shared method where the boundaries of both feed-forward and consideration sublayer with all encoder layers.

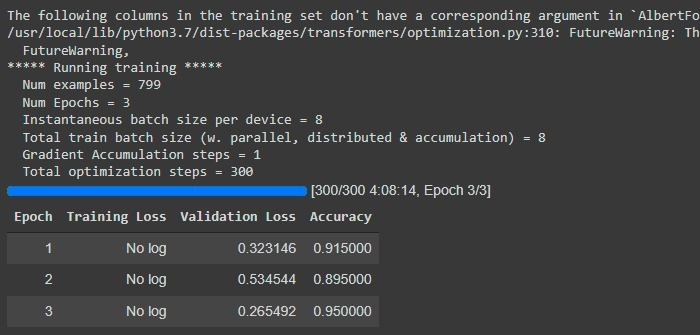
**Factorized embedding layer Parameterization**This is also known as the Reduction technique. In BERT the secret layer embeddings and information layer embeddings are of a similar size. In factorized layer definition the two implanting lattices are isolated. This is on the grounds that BERT utilizes a word piece tokenizer to produce tokens. Word piece tokens are non-relevant, it is gained from the one-hot encoding vectors. The secret layer embeddings require setting subordinate learning. while expanding the size of the secret layer embedding 'H', it will likewise build the size of the word piece inserting 'E'. To stay away from an expansion in the quantity of boundaries the size connection between the secret layer and the Embedding layer is isolated. Where we factorize the word implanting framework into more modest networks. By utilizing these it can diminish the preparation time and deduction season of the BERT model, around 70% of the general number of boundaries are decreased. As we can find in the outline underneath, the ALBERT model has a more modest boundary size contrasted with the comparing BERT mode.

**Test cases:**

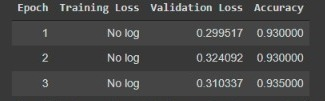
**With 11 epoch:**



**With 3 epochs:**



**With 3 epochs:**



**5.9 ELECTRA**

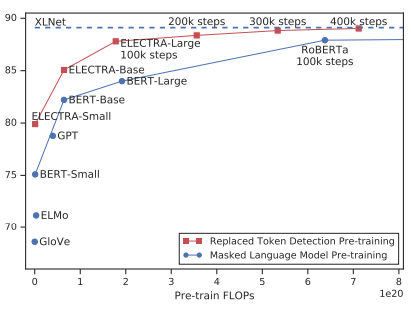
Current state-of-the-art representation learning methods for language can be viewed as learning denoising autoencoders. They select a small subset of the unlabeled input sequence (typically 15%), mask the identities of those tokens (e.g., BERT) or attention to those tokens (e.g., XLNet), and then train the network to recover the original input. While more effective than conventional language-model pre-training due to learning bidirectional representations, these masked language modeling (MLM) approaches incur a substantial compute cost because the network only learns from 15% of the tokens per example.

As an alternative, we use electra for replaced token detection, a pre-training task in which the model learns to distinguish real input tokens from plausible but synthetically generated replacements. Instead of masking, our method corrupts the input by replacing some tokens with samples from a proposal distribution, which is typically the output of a small masked language model.

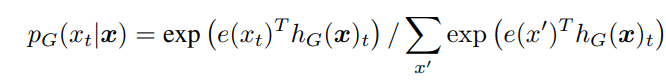
Electra stands for **E**fficiently **L**earning an **E**ncoder that **C**lassifies **T**oken **R**eplacements **A**ccurately. We apply it to pre-train Transformer text encoders that can be fine-tuned on downstream tasks.

Through a series of ablations, we show that learning from all input positions causes ELECTRA to train much faster than BERT. We also show ELECTRA achieves higher accuracy on downstream tasks when fully trained.

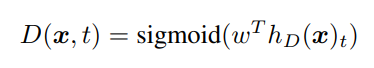
Most current pre-training methods require large amounts of compute to be effective, raising concerns about their cost and accessibility. Since pre-training with more compute almost always results in better downstream accuracies, we argue an important consideration for pre-training methods should be compute efficiency as well as absolute downstream performance. From this viewpoint, we train ELECTRA models of various sizes and evaluate their downstream performance vs. their compute requirement.



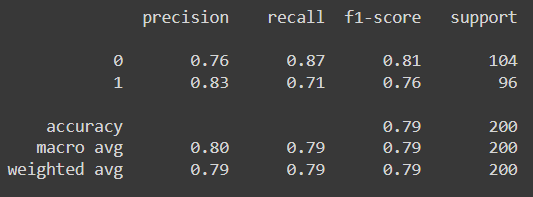
Our approach trains two neural networks, a generator G and a discriminator D. Each one primarily consists of an encoder (e.g., a Transformer network) that maps a sequence on input tokens x = [x1, ..., xn] into a sequence of contextualized vector representations h(x) = [h1, ..., hn]. For a given position t, (in our case only positions where xt = [MASK]), the generator outputs a probability for generating a particular token xt with a softmax layer:



where e denotes token embeddings. For a given position t, the discriminator predicts whether the token xt is “real,” i.e., that it comes from the data rather than the generator distribution, with a sigmoid output layer



**Accuracy with 30 Epoch:**

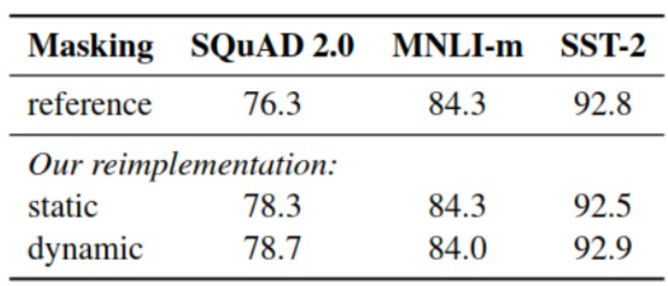


**5.10 ROBERTA**

**5.10.1 Overview**

Roberta stands for Robust **R**obustly **O**ptimized **BERT**-**P**retraining **A**pproach proposed in [Liu et. al.](https://arxiv.org/abs/1907.11692) [1] It is essentially training the Bert model with more data and with different pretraing techniques.

It has been found that BERT performs significantly better when trained on larger datasets. RoBERTa is therefore trained on a huge dataset that contains more than 160GB of uncompressed text.In Roberta Dynamic Masking has been done whereas in BERT the static masking is done.along with the Dynamic masking the pattern for masking the sentence has been provided randomly. The same data set is copied 10 times and provides different masking patterns for the sequences. During the Pretraing the model was trained for 40 epochs 4 epoch for each pattern.[2]



Source[1]

The above image shows a comparison of results from the reference results of the original BERT model.

Tokenization in ROBERTa uses BPE (byte pair Encoding) [3] BPE is a straightforward type of data compression method that substitutes a different byte for the most frequent pair of consecutive bytes in the input data.In line with a subword-based tokenization method, BPE makes sure that the most frequent terms are represented in the vocabulary as a single token while the rare words are divided into two or more subword tokens.[4]

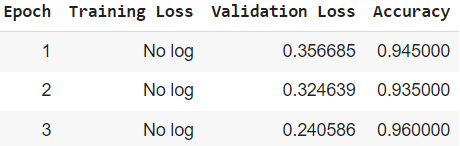
The next thing to ponder is the batch size, the batch size is increased in the Roberta and it helps in running saving the training time.Altogether we can say tht ROBERTa is the same BERT architecture which is trained on large data for longer time without using NSP Objective with BPE( byte pair encoding) along with dynamic masking.

In practice:

So in this project we have used ROBERTa as one of our transfer learning models in order to classify the amazon review to be positive or negative. We have taken reference from the hugging face for the implementation of ROBETa and using pythorch as Framework.

We have taken our dataset and made the changes according to the model input . After that we have tokenized the dataset with the ROBERTa inbuilt tokenizer and iterated it for the whole training dataset. We have loaded the model from hugging faces Pretrained class .

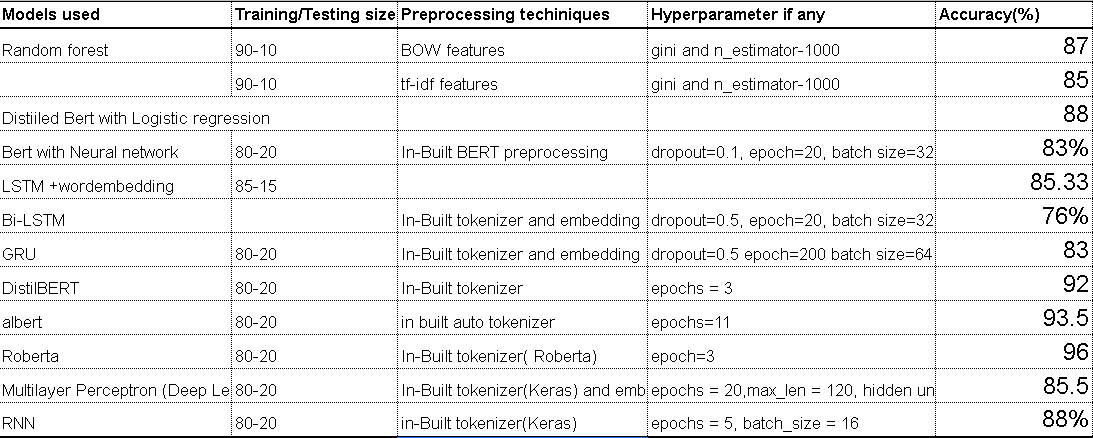
And finally for the training part we are training the model for 3 epochs with the input vector length of 512 .We are using Accuray as our evaluation metric.



So final accuracy is **96%**.

**CHAPTER 6**

**COMPARATIVE STUDY**



From the above table we can infer that Roberta gives us the best results with 96% accuracy. We can also infer that transfer learning models like Roberta, Albert and DistilBert perform much more accurately as compared to machine learning as well as deep learning models.

**REFERENCES:**

**KNN:**

**[1]**[**https://www.analyticsvidhya.com/blog/2021/05/knn-the-distance-based-machine-learning-algorithm/**](https://www.analyticsvidhya.com/blog/2021/05/knn-the-distance-based-machine-learning-algorithm/)

**[2]**[**https://www.tutorialspoint.com/machine\_learning\_with\_python/machine\_learning\_with\_python\_knn\_algorithm\_finding\_nearest\_neighbors.htm**](https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_knn_algorithm_finding_nearest_neighbors.htm)

**[3]** [**https://www.youtube.com/watch?v=4b5d3muPQmA**](https://www.youtube.com/watch?v=4b5d3muPQmA)

**[4]**[**https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761**](https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761)

**ROBERTA**

**[1]** [**https://arxiv.org/abs/1907.11692**](https://arxiv.org/abs/1907.11692)

**[2]**[**https://medium.com/dataseries/roberta-robustly-optimized-bert-pretraining-approach-d033464bd946**](https://medium.com/dataseries/roberta-robustly-optimized-bert-pretraining-approach-d033464bd946)

**[3]**[**https://www.drdobbs.com/a-new-algorithm-for-data-compression/184402829**](https://www.drdobbs.com/a-new-algorithm-for-data-compression/184402829)

**[4]https://towardsdatascience.com/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0**

**Transformers**

**[1]** [**https://jalammar.github.io/illustrated-transformer/**](https://jalammar.github.io/illustrated-transformer/)

**DiltilBERT**

**[1]** [**distilbert-base-cased · Hugging Face**](https://huggingface.co/distilbert-base-cased)

**[2]** [**Fine-tuning a pretrained model — transformers 4.7.0documentation (huggingface.co)**](https://huggingface.co/transformers/v4.8.2/training.html)

**Albert :**

**[1]**[**https://huggingface.co/transformers/v3.1.0/model\_doc/albert.html**](https://huggingface.co/transformers/v3.1.0/model_doc/albert.html)

**[2]** [**https://amitness.com/2020/02/albert-visual-summary/**](https://amitness.com/2020/02/albert-visual-summary/)

**[3]**[**https://www.geeksforgeeks.org/albert-a-light-bert-for-supervised-learning/**](https://www.geeksforgeeks.org/albert-a-light-bert-for-supervised-learning/)

**[4]** [**https://paperswithcode.com/method/albert**](https://paperswithcode.com/method/albert)

**[5]**[**https://huggingface.co/docs/transformers/model\_doc/albert**](https://huggingface.co/docs/transformers/model_doc/albert)