Sentiment Analysis

**Introduction**

Sentiment analysis is contextual mining of text which identifies and extract subjective information in source material, and helping a business to understand the social sentiment of their brand, product or service while monitoring online conversations. However, analysis of social media streams is usually restricted to just basic sentiment analysis and count based metrics. This is akin to just scratching the surface and missing out on those high value insights that are waiting to be discovered. So what should a brand do to capture that low hanging fruit.

These days the data are growing in skyrocketing speed in the website and the trend of online shopping and the showing and the promoting of the products in the website has suddenly increased these days. The big companies like **Netflix, Amazon, Alibaba** has increased the selling of their products in large amount. They use the special type of Algorithm to attract the people so that they visit the Website more times, eventually the traffic of the website increased and finally the selling of the product increase. They collect the data and their sentiment by asking them to rate the certain products, which may be in the form of the Rating or any other text format, then they analyse those data and predict the sentiment of people and for the next time, when they visit the site the similar type of product are recommended to the people. This type of system is applied is applied in almost all kind of Website. The respective company or any other people who are providing the service are tracking us every second and trying to get the data from us and do a similar type of Analysis of our activity.

Let's take a real world example, we use google for solving most of our daily activities, the google is tracking us in every aspect. We notice the google seek for our location and ask for some kind of review or rating, and keeps a log of our activities and for the next time, the similar type of services can be seen on other social media like Facebook, this is the real world experience.

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 the text. Usually, besides identifying the opinion, these systems extract attributes of the expression e.g.:

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

Natural Language Processing (NLP)is a field at the intersection of computer science, artificial intelligence, and linguistics. The goal is for computers to process or “understand” natural language in order to perform various human like tasks like language translation or answering questions.

With the rise of voice interfaces and chatbots, NLP is one of the most important technologies of the 4thIndustrial Revolution and become a popular area of AI. There’s a fast-growing collection of useful applications derived from the NLP field. They range from simple to complex. Below are a few of them:

* Search, spell checking, keyword search, finding synonyms, complex questions answering
* Extracting information from websites such as: products, prices, dates, locations, people or names
* Machine translation (i.e. Google translate), speech recognition, personal assistants *(think about Amazon Alexa, Apple Siri, Facebook M, Google Assistant or Microsoft Cortana)*
* Chat bots/dialog agents for customer support, controlling devices, ordering goods
* Matching online advertisements, **sentiment analysis** for marketing or finance/trading
* Identifying financial risks or fraud

Approaches of Sentiment Analysis

The sentiment analyzer can be built using different methods. Existing approaches to sentiment analyzer can be grouped into three main categories: knowledge-based techniques, statistical methods and deep learning approaches.

Knowledge-based approach

Knowledge-based techniques look for unambiguous affect words such as “good”, “bad”, etc. to determine the sentiment polarity of the text. This is probably the simplest way of implementing a sentiment analyzer but this is also the least accurate because reviews contains complex grammatical structures like negations, idioms, and sarcasms that cannot be all listed explicitly to develop a knowledge-based sentiment classifier. Therefore, knowledge-based approach only works for simple sentences.

Statistical Bag-of-words Methods

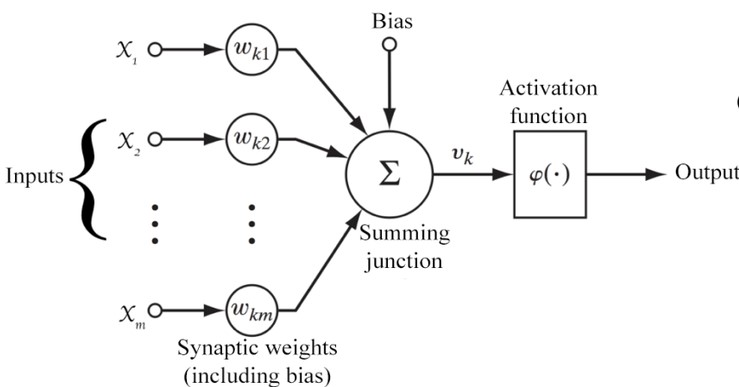
Statistical methods use elements of machine learning such as latent semantic analysis, Support Vector Machines (SVM), bag-of-words (BOW), etc. to calculate the probabilistic polarity of a text. This model is similar to the knowledge-based approach in a way that it also looks at words lexically in isolation, i.e. it analyzes the character composition of words and not the true meaning of the word. For example, lexically, the words “swam” is more similar to “swan” than “swimming”.

The only difference between this BOW method and knowledge-based approach is that in this method, the knowledge is learned from the label dataset whereas in the knowledge-based approach, the fact that word “good” carries positive sentiment and the word “bad” carries negative sentiment is explicitly fed to the algorithm. In the BOW method, the sentiment of the word “good” is learned from the statistics of its appearance in the labeled dataset. For example, if there are 100 positive examples and 100 negative examples in the dataset and 60 positive examples contain the word “good” whereas 20 negative examples contain the word “good”, then the model learns that the word “good” must be 75% positive because out of 80 training examples containing the word “good”, 60 are positive and 20 are negative, and therefore positive equals 0.75 (60/80).

If single word is taken into account at a time, it is called unigram BOW model. The order of the words in the sentence is ignored in such model and important information may be lost. If a pair of consecutive words is taken from the dataset and calculated jointly, then bigram BOW model is obtained. Similarly, if three consecutive words are taken, the trigram BOW model is obtained. By using these n-gram BOW models, the accuracy of the model can be boosted further, but it requires more computer memory and processing resources.

Deep learning approach

Sentiment analysis is a subclass of NLP. Specifically, it is a form of text classification problem where one classifies a piece of text into one of the sentiment classes. The modern approach for NLP is deep learning using Artificial Neural Networks (ANN). ANNs are machine learning models inspired from the structure of the biological brain. An ANN is based on collection of connected units called artificial neurons, analogous to axons in a biological brain. Each connection (synapse) between neurons can transmit a signal to another neuron. The receiving (postsynaptic) neuron can process the signal(s) and then signal downstream neurons connected to it. Neurons may have state, generally represented by real numbers, typically 0 and 1. Neurons and synapses may also have weight that varies as learning proceeds, which can increase or decrease the signal strength as it sends downstream. Further, they may have a threshold such that only if the aggregate signal is below (or above) that level is the downstream signal sent.



Let,

= number of inputs

= vector of input parameters

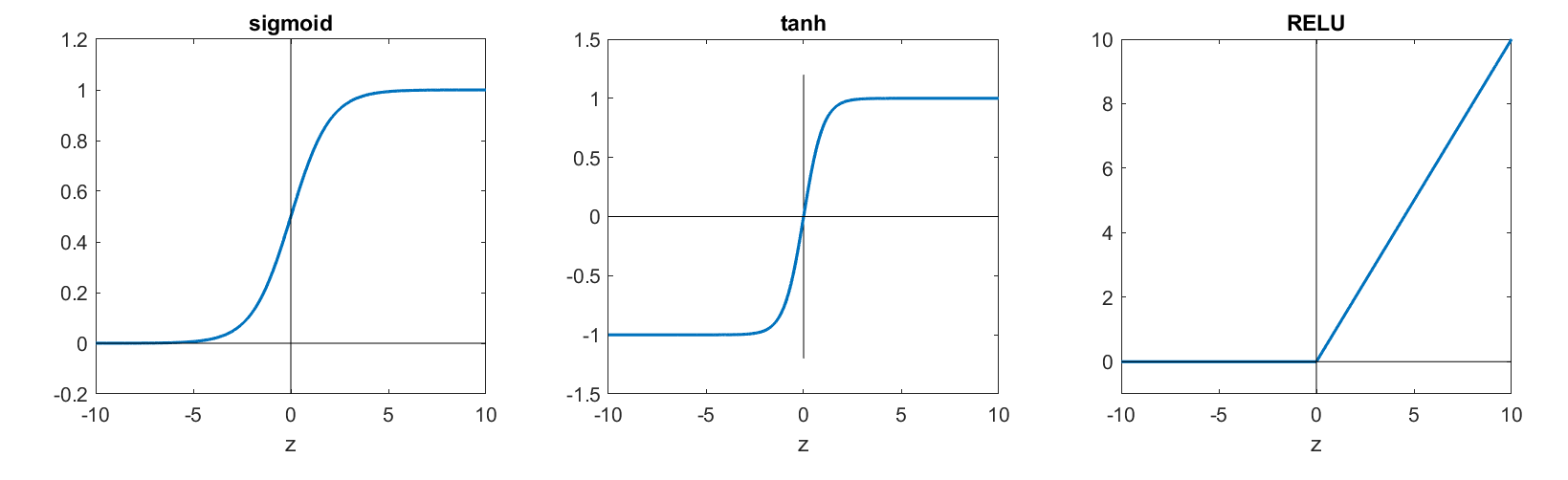
= vector of weights associated with the corresponding inputs

= bias term

Then, the output is,

It can be written in vector form as,

Where, is a monotonically increasing, continuous, differentiable and bounded non-linear function such as:



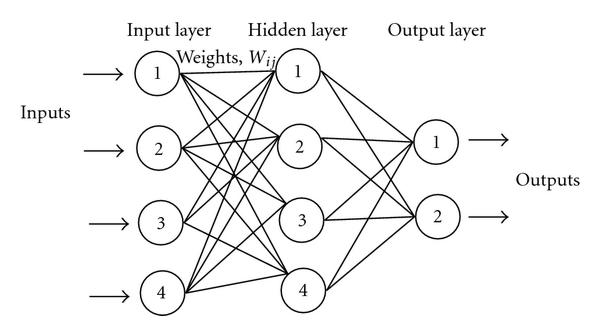


Figure 1.1 (**FIX THIS**) show a simple neural network. All the arrows shown in the figure have weights associated with them. Those weights are adjusted during training phase, which is done using an algorithm called backpropagation. Backpropagation is an algorithm used to calculate the gradient of the cost function with respect to the weights in an ANN. It is commonly used as a part of algorithms that optimize the performance of the network by adjusting the weights. It is also called backward propagation of errors. The cost function is a measure of the error of the neural network model. Most common cost functions are squared error, cross-entropy error, etc. The neural network weights are adjusted to minimize the cost function using any of the optimization algorithms such as gradient descent, BFGS, etc. For instance, in gradient descent, the weight updating step is:

for = 1, 2, 3, ...,

Where,

= number of parameters to learn

= parameter vector to learn

= cost function to minimize

= learning rate

The term “deep learning” basically refers to ANN models having many hidden layers. They could be simple ANNs with multiple hidden layers, or ANNs with special architectures such as Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) and so on.

In deep learning models, there are lots of parameters to learn (adjust) and usually the training set is not sufficient to learn those parameters with great certainty. Therefore, deep learning models are prone to overfitting. Hence, regularizing the cost function is vital for good performance of the model.

**Background**

For this project we have also used the similar kind of the approach for the Book Recommendation System. The recommendation system consists of mainly 4 parts, One of them is the sentiment Analysis for the Text review. The English text review from the customer/user is analysed so that the sentiment of the respective can be acquired for the respective books, we can infer, about the customer thoughts toward to book and the overall rating is done, by considering the number of times of clicking the book and the starring of the book by the user. When the end user gives the certain review about the book then the certain Deep Learning Algorithms are applied to the review so the corresponding rating is generated as the output of the model. This section is mainly for the different approaches used by us during the analysis of the user text review. This sentiment analysis part does not only rely on book, this can be used for every aspect of the product. This can be used for any other system, like Restaurant, Hotels, Party Palaces, Tourist Destination or any other commercial product where there is a provision of giving review for that item. This project mainly focuses on the Book Recommendation System, so we are using the sentiment score of the text input to evaluate the overall sentiment of Any Book that the user gives/uses.

**Data Sources**

The data are the main part of any project. For the successful completion of the project the data, facts and other information bolster the outcome and the accuracy of the project. The whole project is of data driven approach in which we use data for discovering new knowledge from the existing one. If we have some data and information about something then we can infer or predict the outcome of the system for the new attributes. Similar kind of approach is applied in our project. The deep learning models for the generation of sentiment score of the text, we have used the Movie Datasets provided by the IMDd. The Deep Learning Framework PyTorch Provides the access of the IMDb datasets for the educational purpose, hence we have used for our training model

1. **Review Dataset v1.0**

**Overview**

This dataset contains movie reviews along with their associated binary sentiment polarity labels. It is intended to serve as a benchmark for sentiment classification. This document outlines how the dataset was gathered, and how to use the files provided

**Dataset**

The core dataset contains 50,000 reviews split evenly into 25k train and 25k test sets. The overall distribution of labels is balanced (25k pos and 25k neg). We also include an additional 50,000 unlabeled documents for unsupervised learning.

This is one of the biggest labeled movie review datasets available and hundreds of researches on sentiment analysis and NLP have been carried out that use this datasets. But one of the main problems with this dataset is that it only contains highly polar reviews for training and testing, i.e the reviews in this datasets are either very positive, or very negative. There are no slightly positive, slightly negative, or neutral movie reviews. So this datasets, while being useful for the binary sentiment classification tasks, is skewed and not very effective for the fine-grained sentiment classification task like in this project where it is required to classify a review into a rating scale from 1 to 5.

2.**GloVe: Global Vectors for Word Representation**

**GloVe**, coined from Global Vectors, is a model for distributed word representation. The model is an unsupervised learning algorithm for obtaining vector representations for words. This is achieved by mapping words into a meaningful space where the distance between words is related to semantic similarity.Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. It is developed as an open-source project at Stanford. As log-bilinear regression model for unsupervised learning of word representations, it combines the features of two model families, namely the global matrix factorization and local context window method. There are different types of the GloVe models for the word representations.

Here, we'll be using the "**glove.6B.100d**" vectors". glove is the algorithm used to calculate the vectors, go here for more. 6B indicates these vectors were trained on 6 billion tokens and 100d indicates these vectors are 100-dimensional

The theory is that these pre-trained vectors already have words with similar semantic meaning close together in vector space, e.g. "terrible", "awful", "dreadful" are nearby. This gives our embedding layer a good initialization as it does not have to learn these relations from scratch.

**Data Preparation**

One of the main concepts of TorchText is the Field. These define how your data should be processed. In our sentiment classification task the data consists of both the raw string of the review and the sentiment, either "pos" or "neg".

The parameters of a Field specify how the data should be processed.

We use the TEXT field to define how the review should be processed, and the LABEL field to process the sentiment.

Our TEXT field has tokenize='spacy' as an argument. This defines that the "tokenization" (the act of splitting the string into discrete "tokens") should be done using the spaCy tokenizer. If no tokenize argument is passed, the default is simply splitting the string on spaces.

LABEL is defined by a LabelField, a special subset of the Field class specifically used for handling labels. We will explain the *dtype* argument later.

The IMDb dataset only has train/test splits, so we need to create a validation set. We can do this with the .split() method.

By default this splits 70/30, however by passing a split\_ratio argument, we can change the ratio of the split, i.e. a split\_ratio of 0.8 would mean 80% of the examples make up the training set and 20% make up the validation set.

We also pass our random seed to the random\_state argument, ensuring that we get the same train/validation split each time.

**Data Preprocessing**

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing. This is the basic after the collection of the data. The user may not get the desired datasets for his need, so the user needs to modify the datasets according the the need and demand of the project. During this course, the user may need to handle the missing values, outliers or any other mis-representation of the datasets.

We had also faced such disabilities after the collection of datasets from the amazon. We needed only the words that represent the real meaning, but the text review contains a large number of words including regular expression, punctuation, commas, full stops and other forms of the words like past or future form, which are not needed and they are redundant to our project. For this we used NLTK python library for Natural Language processing.

**Data Cleaning**

Data cleansing or data cleaning is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data. Since the data obtained from the IMDB movie datasets were already built, there were no any faults to be cleaned.

**Data Reduction**

Data reduction is the transformation of numerical or alphabetical digital information derived empirically or experimentally into a corrected, ordered, and simplified form. The basic concept is the reduction of multitudinous amounts of data down to the meaningful parts.

Data reduction can be achieved several ways. The main types are data deduplication, compression and single-instance storage. Data deduplication, also known as *data dedupe*, eliminates redundant segments of data on storage systems. It only stores redundant segments once and uses that one copy whenever a request is made to access that piece of data. Data dedupe is more granular than single-instance storage. Single-instance storage finds files such as email attachments sent to multiple people and only stores one copy of that file. As with dedupe, single-instance storage replaces duplicates with pointers to the one saved copy.

**Normalization**

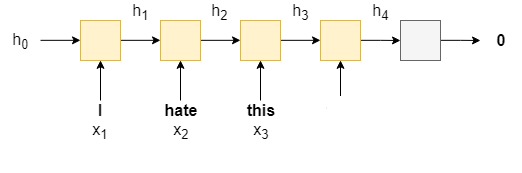
Data normalization is the process of converting the data to any kind of canonical forms. In terms of the machine learning or signal processing, data normalization refers to transforming the values to al limited range or scale. For example, if some movie review data de[rovides sentiment labels as rating scales from 1 to 10, it is required to normalize that to convert the labels to the required value that is 1 to 5 . In this case, a simple division by 2 seems to be sufficient but on other cases, it might take more such operations such as subtracting mean value and then dividing by the standard deviation.

**Tools and Technologies**

Extracting the sentiment of the text is really tough job, but the availability of the different kinds of modern hardware and software and Artificial Intelligence technologies we have successfully overcome that barrier and able to extract the sentiment of the text review which we give as use input for the product, being specific for this project:BOOK

Simple Sentiment Analysis

They are the form of the Artificial Neural Network, which are mainly used in sequence modelling for in Natural Language Processing and other different kinds of speech and text processing methodologies. The recurrent neural networks have to property of saving the previous information of the sequences so it has made possible to do the Speech Recognition and Understanding job. These frameworks can be used for other various types of application like getting the polarity of a sentence and finding the sentiment score of a text.



We'll be using a recurrent neural network (RNN) as they are commonly used in analysing sequences. An RNN takes in sequence of words, *X =* {x1,...,xT} , one at a time, and produces a hidden state, h , for each word. We use the RNN recurrently by feeding in the current word xt  as well as the hidden state from the previous word, ht−1 , to produce the next hidden state, ht .

ht=RNN(xt,ht−1)

Once we have our final hidden state, hT , (from feeding in the last word in the sequence, xT ) we feed it through a linear layer, f , (also known as a fully connected layer), to receive our predicted sentiment, y^ = f(hT) .

# Updated Sentiment Analysis

The output of the simple sentiment analysis was very poor, it could classify the sentences with less than 50% with was very poor. As a solution we made the RNN architecture a little bit complex by adding following parameters.

**1.Packed padded sequences**

We'll be using *packed padded sequences*, which will make our RNN only process the non-padded elements of our sequence, and for any padded element the output will be a zero tensor. To use packed padded sequences, we have to tell the RNN how long the actual sequences are.

**2.Pre-trained word embeddings: GloVe**

Pre-trained word embedding help to align the input word vector in the vector of words space, in which the similar kinds of words are aligned in the similar dimension of word space. For this we have used. The statistics of word occurrences in a corpus is the primary source of information available to all unsupervised methods for learning word representations, and although many such methods now exist, the question still remains as to how meaning is generated from these statistics, and how the resulting word vectors might represent that meaning.

The motivation for GloVe starts with that *ratios* between the probabilities of words appearing next to each other carry more information than considering those probabilities individually. Don’t worry if this doesn’t make sense. I’ll break it down. Let us assume (i,j)element of the co-occurrence matrix X, X_{i,j} denotes the number of times words j occur near i. Next let us define P_{ij}=P(j|i)=X_{ij}/\sum_{\forall k}X_{ik}. These are pretty basic things.

With this notation we can find why the ratio between probabilities work. Consider 4 words; *solid, gas, water* and *fashion*. If you calculate the ratio P_{ice,k}/P_{steam,k} where k \in \{solid, gas, water, fashion\} from X, you will make following observations.

1. P_{ice,solid}/P_{steam,solid} will be very high
2. P_{ice,gas}/P_{steam,gas} is very low
3. P_{ice,water}/P_{steam,water} will be higher than P_{ice,fashion}/P_{steam,fashion}

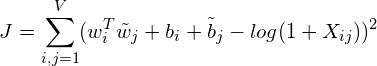
## 

## Deriving the Cost Function

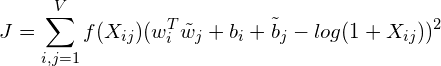
So it can be seen that this ratio is quite expressive. Now we say that whatever we are to derive should start with this ratio, leading to the function below.

\begin{equation*} F(w_i,w_j,\tilde{w}_k)=\frac{P_{ik}}{P_{jk}} \end{equation*}

Then after some serious mathematical crunching the authors arrive at the following cost function.



where V is the vocabulary size, w_i is the target word, \tilde{w}_j is the context word and b_i is the bias for word w_i. But this cost function treat words far-apart and close together equally. To solve this, we introduce a weighing function, turning the equation to,



You can choose any function for f that satisfy a set of properties specified in the paper. The recommended function for f is as below.



This is it. The main contribution of the paper is this new cost function we just saw. Then you train the model as you would do with skip-gram but with the new cost function.

## 

## Big Picture

This is how it looks like when everything comes together. Inputs are w_is, Outputs are \tilde{w}_js, bias embeddings are b_i and \tilde{b}_js. Note that the cost function is defined only for a single input,output set. But the picture depicts the algorithm for a batch of data of size b.

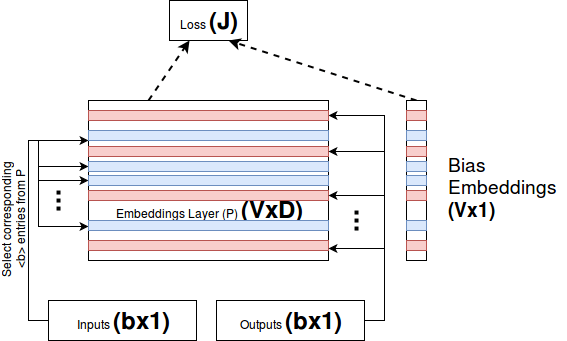
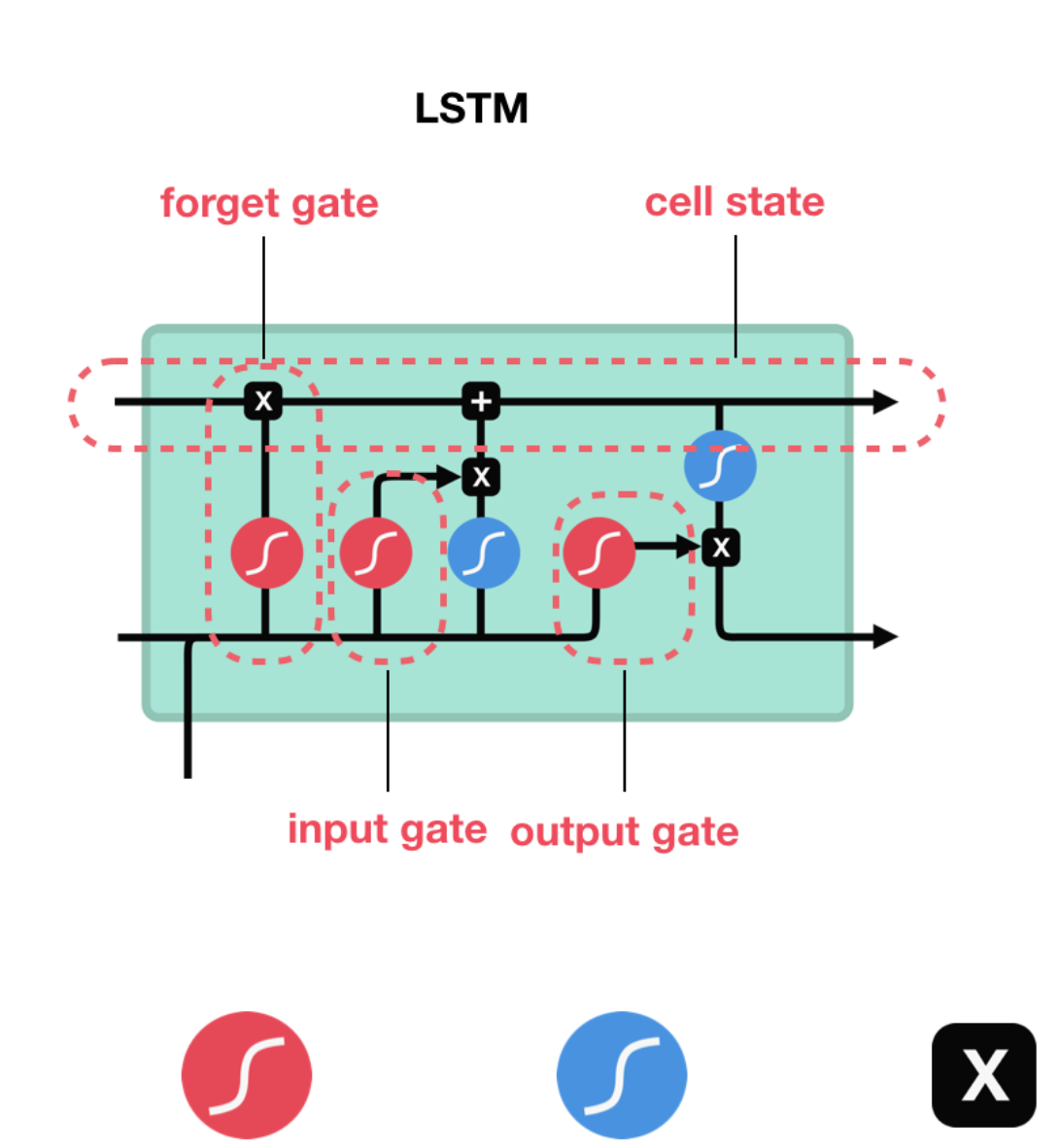


Figure :GloVe, High-level Architecture

**3.Different RNN architecture**

We'll be using a different RNN architecture called a Long Short-Term Memory (LSTM). Standard RNNs suffer from the vanishing gradient problem. LSTMs overcome this by having an extra recurrent state called a cell, c - which can be thought of as the "memory" of the LSTM - and the use use multiple gates which control the flow of information into and out of the memory. We can simply think of the LSTM as a function of  **xt** , **ht** and **ct** , instead of just **xt**  and  **ht** .

(ht,ct) = LSTM(xt,ht,ct)

The initial cell state, **c0** , like the initial hidden state is initialized to a tensor of all zeros. The sentiment prediction is still, however, only made using the final hidden state, not the final cell state, i.e. y^=f(hT) .

Figure: Basic LSTM cell

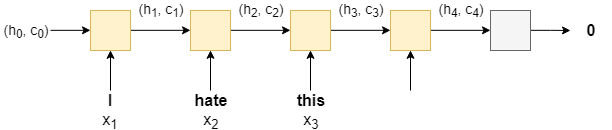


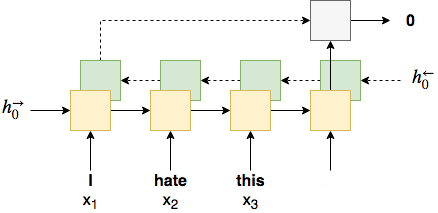
Figure: Cascaded LSTM network

**Bidirectional RNN**

The concept behind a bidirectional RNN is simple. As well as having an RNN processing the words in the sentence from the first to the last (a forward RNN), we have a second RNN processing the words in the sentence from the last to the first (a backward RNN). At time step t , the forward RNN is processing word xt , and the backward RNN is processing word xT−t+1.

We make our sentiment prediction using a concatenation of the last hidden state from the forward RNN (obtained from final word of the sentence), h→T , and the last hidden state from the backward RNN (obtained from the first word of the sentence), h←T , i.e. y^=f(h→T,h←T)

The image below shows a bi-directional RNN, with the forward RNN in orange, the backward RNN in green and the linear layer in silver.multi-layer RNN



**Regularization**

Although we've added improvements to our model, each one adds additional parameters. Without going into overfitting into too much detail, the more parameters we have in our model, the higher the probability that your model will overfit (memorize the training data, causing a low training error but high validation/testing error, i.e. poor generalization to new, unseen examples). To combat this, we use regularization. More specifically, we use a method of regularization called dropout. Dropout works by randomly dropping out (setting to 0) neurons in a layer during a forward pass. The probability that each neuron is dropped out is set by a hyperparameter and each neuron with dropout applied is considered independently. One theory about why dropout works is that a model with parameters dropped out can be seen as a "weaker" (less parameters) model. The predictions from all these "weaker" models (one for each forward pass) get averaged together within the parameters of the model. Thus, your one model can be thought of as an ensemble of weaker models, none of which are over-parameterized and thus should not overfit.

**A different Optimizer**

The only change we'll make here is changing the optimizer from SGD to Adam. SGD updates all parameters with the same learning rate and choosing this learning rate can be tricky. Adam adapts the learning rate for each parameter, giving parameters that are updated more frequently lower learning rates and parameters that are updated infrequently higher learning rates.

**How Does the Deep Learning Model(RNN+LSTM) able to Classify the Sentences?**

We can now use our model to predict the sentiment of any sentence we give it. As it has been trained on book reviews, the sentences provided should also be book reviews.

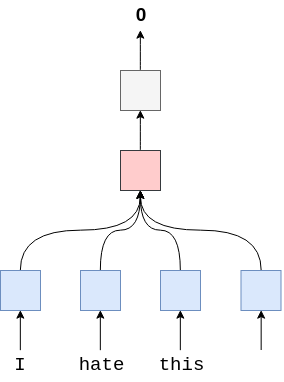
When using a model for inference it should always be in evaluation mode. If this tutorial is followed step-by-step then it should already be in evaluation mode (from doing **evaluate** on the test set), however we explicitly set it to avoid any risk.

Our **predict\_sentiment** function does a few things:

* sets the model to evaluation mode
* tokenizes the sentence, i.e. splits it from a raw string into a list of tokens
* indexes the tokens by converting them into their integer representation from our vocabulary
* gets the length of our sequence
* converts the indexes, which are a Python list into a PyTorch tensor
* add a batch dimension by **unsqueezeing**
* converts the length into a tensor
* squashes the output prediction from a real number between 0 and 1 with the **sigmoid** function
* converts the tensor holding a single value into an integer with the item() method

We are expecting reviews with a negative sentiment to return a value close to 0 and positive reviews to return a value close to 1

**Faster Sentiment Analysis**

In the updated sentiment analysis we were able to have pretty good accuracy of about 84%. After further study, came to know that it can be implemented and accuracy can be increased with FastText model from the paper “Bag of Tricks for Efficient Text Classification”. The model is much more less and with fewer parameters, but having greater accuracy which is good for both time and space complexity. The details of the hyperparameter used will be showed in the Experiment and the Outcome section. 

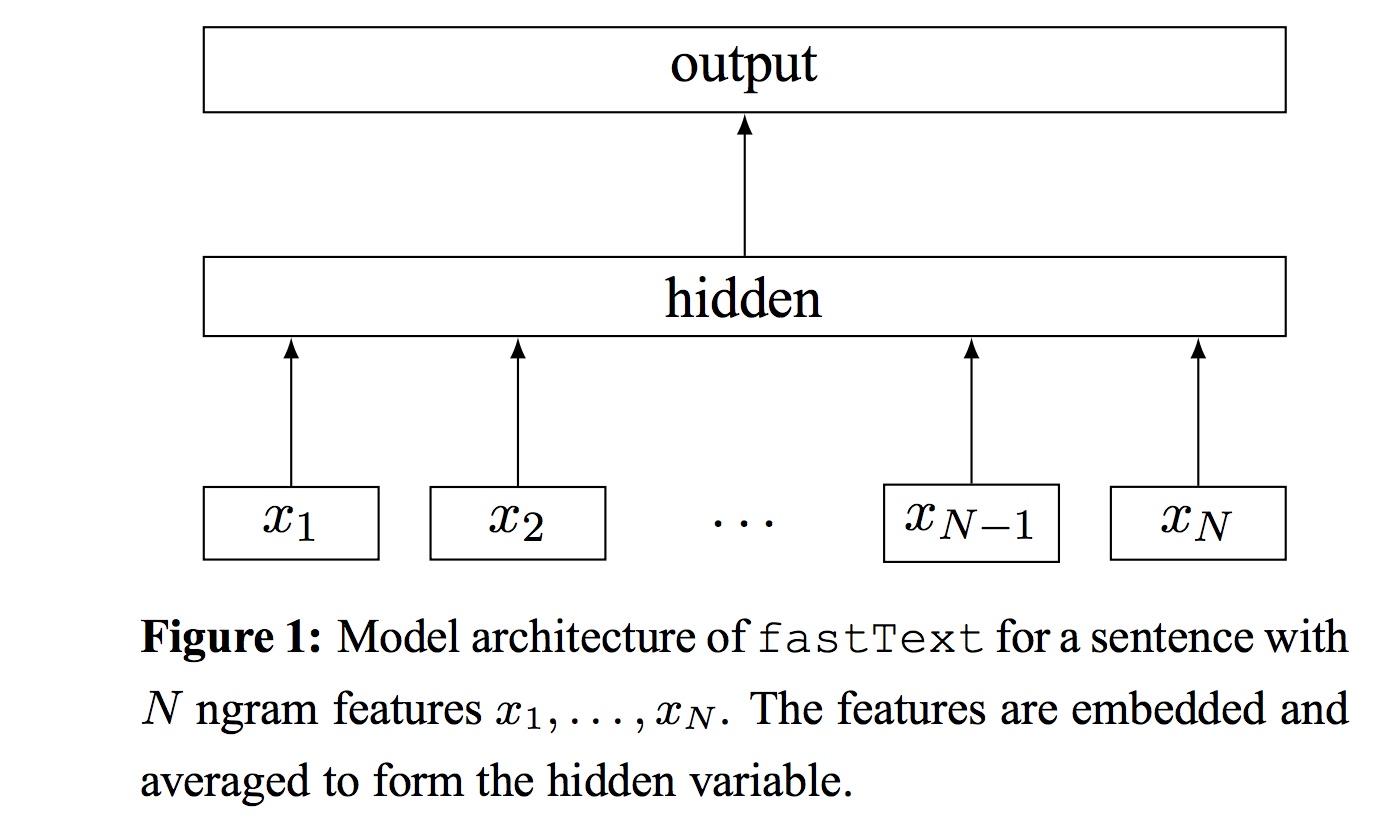
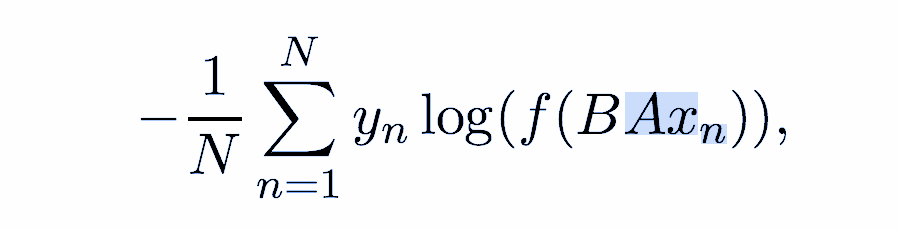


Figure : Model Architecture of *FastText*

The negative log likelihood is minimized with the following function

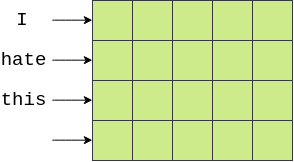


# **Convolutional Sentiment Analysis**

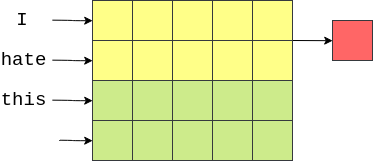
Traditionally, CNNs are used to analyse images and are made up of one or more *convolutional* layers, followed by one or more linear layers. The convolutional layers use filters (also called *kernels* or *receptive fields*) which scan across an image and produce a processed version of the image. This processed version of the image can be fed into another convolutional layer or a linear layer. Each filter has a shape, e.g. a 3x3 filter covers a 3 pixel wide and 3 pixel high area of the image, and each element of the filter has a weight associated with it, the 3x3 filter would have 9 weights. In traditional image processing these weights were specified by hand by engineers, however the main advantage of the convolutional layers in neural networks is that these weights are learned via backpropagation.

The intuitive idea behind learning the weights is that our convolutional layers act like *feature extractors*, extracting parts of the image that are most important for your CNN's goal, e.g. if using a CNN to detect faces in an image, CNN may be looking for features such as the existence of a nose, mouth or a pair of eyes in the image.

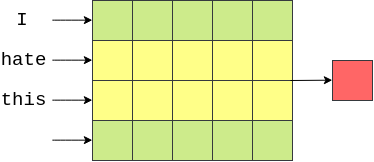
So why use CNNs on text? In the same way that a 3x3 filter can look over a patch of an image, a 1x2 filter can look over a 2 sequential words in a piece of text, i.e. a bi-gram. In this CNN model we will instead use multiple filters of different sizes which will look at the bi-grams (a 1x2 filter), tri-grams (a 1x3 filter) and/or n-grams (a 1xn filter) within the text.



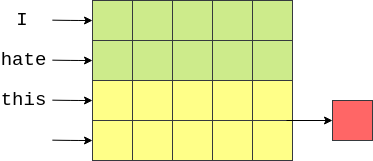
The first major hurdle is visualizing how CNNs are used for text. Images are typically 2 dimensional whereas text is 1 dimensional. This is how we can visualize our words in 2 dimensions, each word along one axis and the elements of vectors across the other dimension. Consider the 2 dimensional representation of the embedded sentence given.

We can then use a filter that is **[n x emb\_dim]**. This will cover n sequential words entirely, as their width will be emb\_dimdimensions. Consider the image below, with our word vectors are represented in green. Here we have 4 words with 5 dimensional embeddings, creating a [4x5] "image" tensor. A filter that covers two words at a time (i.e. bi-grams) will be **[2x5]** filter, shown in yellow, and each element of the filter with have a *weight* associated with it. The output of this filter (shown in red) will be a single real number that is the weighted sum of all elements covered by the filter.

The filter then moves "down" the image (or across the sentence) to cover the next bi-gram and another output (weighted sum) is calculated.



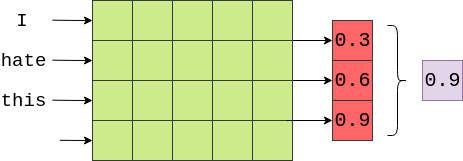
Finally, the filter moves down again and the final output for this filter is calculated



In our case (and in the general case where the width of the filter equals the width of the "image"), our output will be a vector with number of elements equal to the height of the image (or length of the word) minus the height of the filter plus one,

4−2+1=3 in this case.

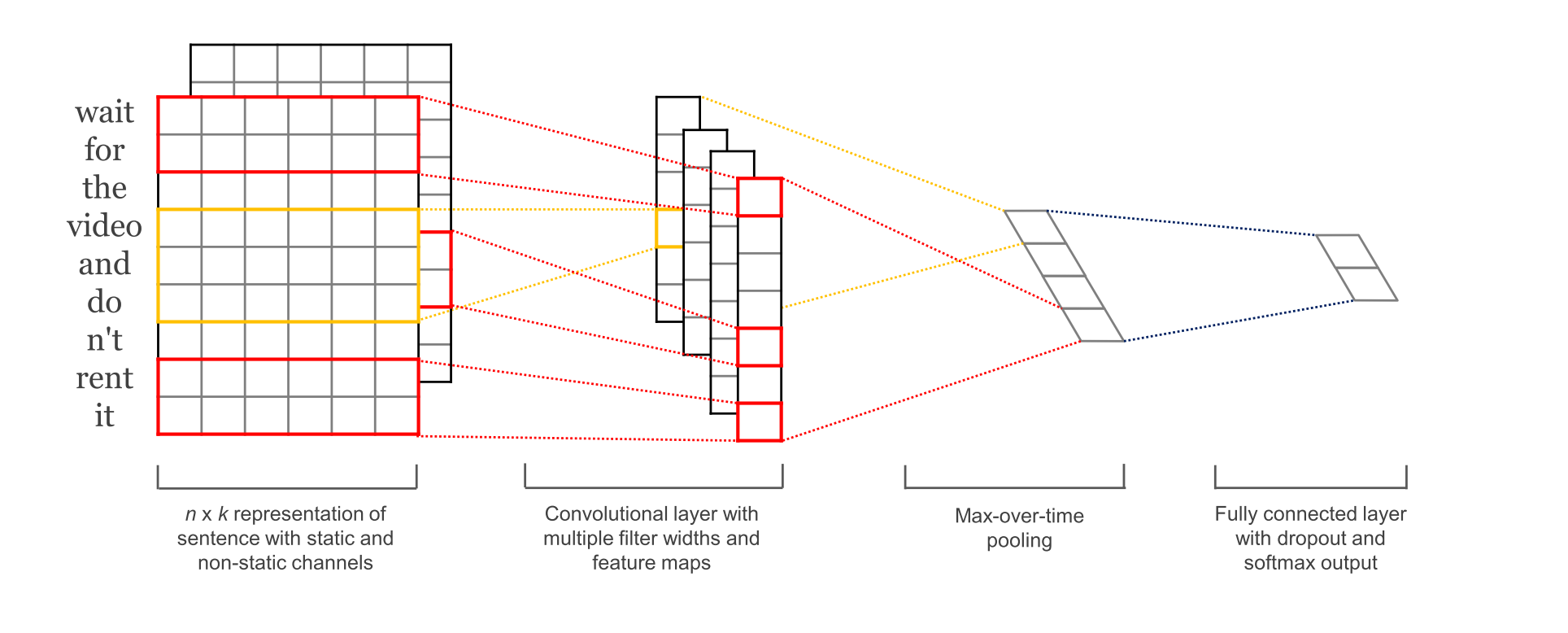
This example showed how to calculate the output of one filter. Our model (and pretty much all CNNs) will have lots of these filters. The idea is that each filter will learn a different feature to extract. In the above example, we are hoping each of the **[2 x emb\_dim]** filters will be looking for the occurence of different bi-grams.

In our model, we will also have different sizes of filters, heights of 3, 4 and 5, with 100 of each of them. The intuition is that we will be looking for the occurence of different tri-grams, 4-grams and 5-grams that are relevant for analysing sentiment of movie reviews.

The next step in our model is to use *pooling* (specifically *max pooling*) on the output of the convolutional layers. This is similar to the FastText model where we performed the average over each of the word vectors, implemented by the F.avg\_pool2d function, however instead of taking the average over a dimension, we are taking the maximum value over a dimension. Below an example of taking the maximum value (0.9) from the output of the convolutional layer on the example sentence (not shown is the activation function applied to the output of the convolutions).

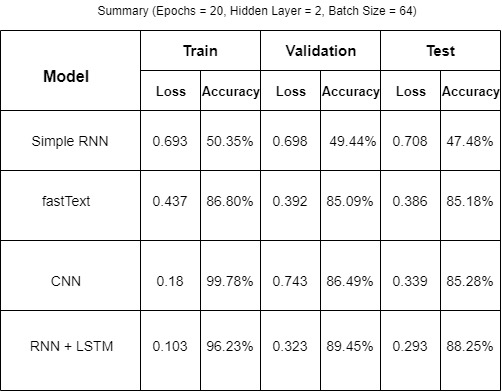
The idea here is that the maximum value is the "most important" feature for determining the sentiment of the review, which corresponds to the "most important" n-gram within the review. How do we know what the "most important" n-gram is? Luckily, we don't have to! Through backpropagation, the weights of the filters are changed so that whenever certain n-grams that are highly indicative of the sentiment are seen, the output of the filter is a "high" value. This "high" value then passes through the max pooling layer if it is the maximum value in the output.

As our model has 100 filters of 3 different sizes, that means we have 300 different n-grams the model thinks are important. We concatenate these together into a single vector and pass them through a linear layer to predict the sentiment. We can think of the weights of this linear layer as "weighing up the evidence" from each of the 300 n-grams and making a final decision.



**Experiments and Outcomes**

After the data collection and its preprocessing the main tough job was to train and test the Deep learning model. We didn’t get accuracy by using the single model so had to test many models, which is by using different Neural Network Models, explained earlier. Different types of models were trained and validated and tested on the datasets and were compared. According to the performance of the model. The model with highest accuracy was used in the Project. The experiment was tested with simple RNN cell and was upgraded to Advanced RNN architectures to avoid the different kinds of hurdles. During the experiment in the Simple RNN cell the overall accuracy was only 47% which was very very poor and couldn’t classify the sentences properly. The next move was use of Advanced RNN architectures using LSTM and GRU which helped in solving the problem of Exploding Gradient Problem by removing Long Term Dependencies. After doing this the Overall Accuracy was Increased to 88%, which was a great breakthrough for the Deep Learning Model. CNN had also great performance and was able to classify the sentence with 85% accuracy. The following table shows the comparative analysis of the performance of the different models.



The following plot shows the accuracy and loss of the deep learning classifier which was trained to classify the sentences. Due to very poor performance d of the Simple RNN model we have not considered that model, neither used that in the model. The performance was very good in RNN and CNN so the model was considered.

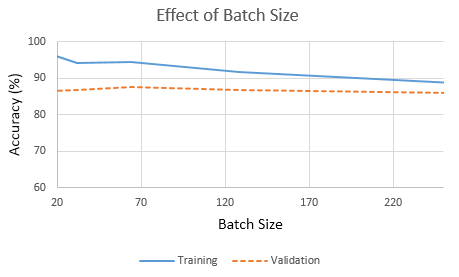
**Upgraded RNN model (RNN+LSTM)**

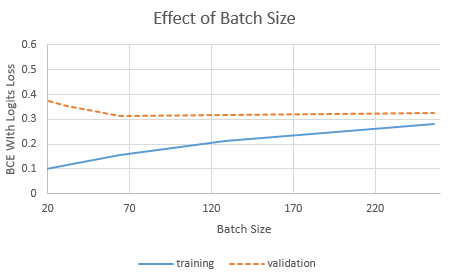
For the LSTM model different hyperparameter were applied to observe the performance of the model in terms of the accuracy and loss. There are three partitions of the datasets. Training set, Test Set and Validation set. The model was trained in the Training set, was validated with validation set and tested on the test set. The model provided different outcomes for different hyperparameter. The test accuracy was better in case of the RNN+LSTM model hence, this model was used in the project and was considered for analysis for change of **hyperparameter**.

**Effect Due to Batch Size**

The **batch size** defines the number of samples that will be propagated through the network.

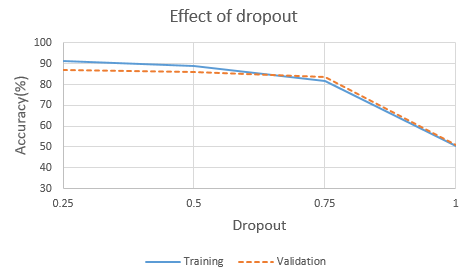
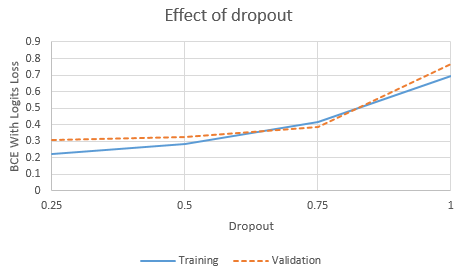
For instance, let's say you have 1050 training samples and you want to set up a batch\_size equal to 100. The algorithm takes the first 100 samples (from 1st to 100th) from the training dataset and trains the network. Next, it takes the second 100 samples (from 101st to 200th) and trains the network again. We can keep doing this procedure until we have propagated all samples through the network. Problem might happen with the last set of samples. In our example, we've used 1050 which is not divisible by 100 without remainder. The simplest solution is just to get the final 50 samples and train the network.

The accuracy and Loss of the RNN model due to the effect of the batch size shows below.



**Effect of Dropout**

It may not be obvious that, aways the model fit properly, sometimes due to very high number of neurons, the model may get overfit ot underfit, which is undesirable in the Machine Learning. The overfitting takes time to fit the model and yet not able to classify the text properly. Due to the underfitting, it is sure that it wouldn't classify the input properly. So some of the neurons in neural network must be dropped out and hence make no use of that. Dropout is a technique where randomly selected neurons are ignored during training. They are “dropped-out” randomly. This means that their contribution to the activation of downstream neurons is temporarily removed on the forward pass and any weight updates are not applied to the neuron on the backward pass. The dropout number is a probability that ranges from 0-1.

The plot for the accuracy of the training with different dropout probability is shown below

**Losses and Accuracy Plots**

Now it's time to visualize the graphs, how the accuracy and losses were changes during the training process. Generally speaking, the training loss, validating losses gradually decrease over the number of iterations and corresponding accuracy increases. We had also obtained the similar kind of results. The nominal and optimal hyperparameters were found to be the following and following those data we had observed the accuracy and losses. After doing a lot of training and testing the Following Hyperparameter were found to have the capability of best fitting our Deep Learning Model

Number of Epochs : 20

Batch Size : 64

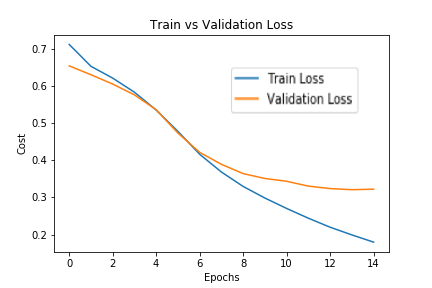
Number of Filters : 100

Optimization Algorithm: Adam

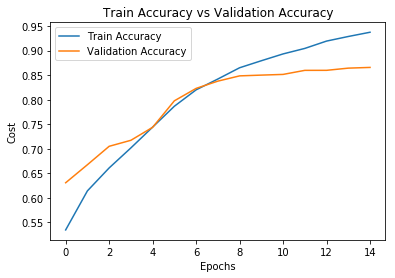
Learning Rate : 0.001 (default for Adam)

Momentum : 0.9 (default for Adam)

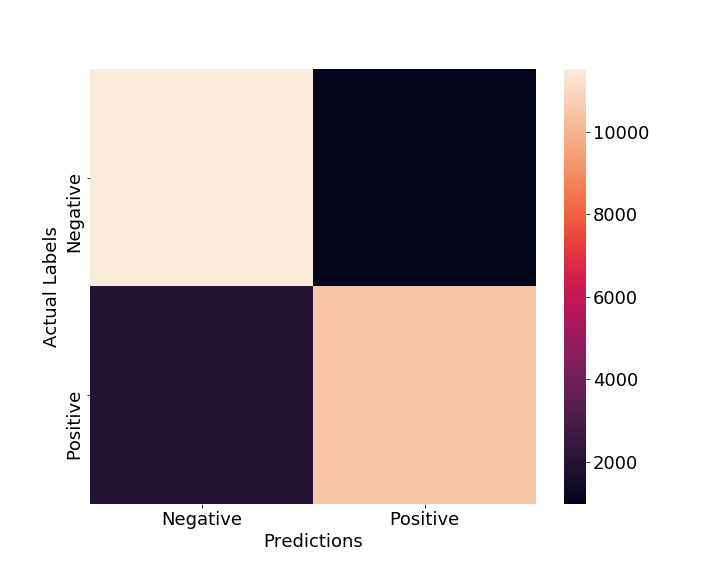
Observing the Training Loss for the Training Set we observed the following graph. Which shows the loss is gradually decreasing over Number of Epochs



Similarly for validation Set, the similar graph was plotted and obtained the following graph. The plot of the validation accuracy of the graph looked like



**Confusion Matrix**

In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix. A confusion matrix is a table that is often used to describe the performance of a classification model (or “classifier”) on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm. It allows easy identification of confusion between classes e.g. one class is commonly mislabeled as the other. Most performance measures are computed from the confusion matrix.

We have plotted the confusion matrix for our test data sets for RNN\_LSTM model. Which shows the number of sentences, which were classified truly or badly. The pure negative sentence is considered as 0 and pure negative sentence is considered as 1. The positivity of any sentence increased as the softmax output increases from 0 to 1. The below figure shows details of the sentences classified.

From the calculation it is found that,

False Negative = 11016

True Positive = 10033

False Positive = 1184

False Negative = 2467

= 49.49%

= 49.49%

= 87.17%

= 47.66%

= 90.5%