CHAPTER 1 – INTRODUCTION

1.1 Overview of Problem Area

Today we know that machines have become smarter than us and can help us with every aspect of life, the technologies have reached to an extent where they can do all the tasks of human beings like household tasks, controlling home devices, making appointments etc. The field which makes these things happen is Machine Learning. Machine Learning trains the machines with some data which makes it capable of acting when tested by the similar type of data. The machines have become capable of understanding human languages using Natural Language Processing. Today researches are being done in the field of text analytics.

As the project title suggests, Text Summarizer is an application which helps in summarizing the text. We can upload our data and this application gives us the summary of that data in as many numbers of lines as we want. The product is mainly a text summarizing using Deep Learning concepts. The main purpose is to provide reliable summaries of web pages or uploads files depend on the user's choice. The unnecessary sentences will be discarded to obtain the most important sentences.

Text summarization methods are greatly needed to address the ever-growing amount of text data available online to both better help discover relevant information and to consume relevant information faster. There is a great need to reduce much of this text data to shorter, focused summaries that capture the salient details, both so we can navigate it more effectively as well as check whether the larger documents contain the information that we are looking for.

Based on language, there are three kinds of summaries: multi-lingual, mono-lingual and cross-lingual summaries. When language of source and target document is same, it’s a mono-lingual summarization system. When source document is in many languages like English, Hindi, Punjabi and summary is also generated in these languages, then it is termed as a multi-lingual summarization system. If source document is in English and the summary generated is in Hindi or any other language other than English, then it is known as a cross-lingual summarization system. Another common type is Web-based summarization. Nowadays users are facing information in abundance on the internet. Web pages on internet are doubling every year. Some search engines like Google Fast, Alta Vista, etc help users to find the information they require but they return a list of substantial number of web pages for a single query. As a result, users need to go through multiple pages to know which documents are relevant and which are not and most of the users give up their search in the first try. Therefore, web based summaries summarize valuable information present in the web pages. E-mail based summarization is also a type of summarization in which email conversations are summarized. Email has become an effective way of communication because of its high delivery speed and lack of cost. Emails keep on coming in the inbox due to which email overloading problem occurs and considerable time is spent in reading, sorting and archiving the incoming emails.

1.2 Why Text Summarizer ?

In the modern Internet age, textual data is ever increasing; we need some way to condense this data while preserving the information and meaning. Text summarization is a fundamental problem that we need to solve. I would help in easy and fast retrieval of information, and use the retrieve information for our required purpose.

Propelled by the modern technological innovations, data is to this century what oil was to the previous one. Today, our world is parachuted by the gathering and dissemination of huge amounts of data.In fact, the International Data Corporation (IDC) projects that the total amount of digital data circulating annually around the world would sprout from 4.4 zettabytes in 2013 to hit 180 zettabytes in 2025.With such a big amount of data circulating in the digital space, there is need to develop machine learning algorithms that can automatically shorten longer texts and deliver accurate summaries that can fluently pass the intended messages.Furthermore, applying text summarization reduces reading time, accelerates the process of researching for information, and increases the amount of information that can fit in an area.

1.3 Problem Specification

“Creating short, accurate, and fluent summaries from larger text documents using Machine Learning”.

The trainable Text Summarizer is expected to learn the patterns which lead to the summaries, by identifying relevant feature values which are most correlated with the classes “correct” or “incorrect”. When a new document is given to the system, the “learned” patterns are used to classify each sentence of that document into either a “correct” or “incorrect” sentence, producing an extractive summary. A crucial issue in this framework is how to obtain the relevant set of features; the next section treats this point in more detail.

Researchers and students constantly face the scenario where it’s impossible to read most if not all of the newly published papers to be informed of latest progress and when the work on a research project, the time spent on reading literature review seems endless. The goal of this project is to design a Summarizer that generates a summary that helps the text to read in short time and focus on key features. The most important task is to generate an efficient scoring algorithm that would produce the best results for a wide range of text types. The only means to arrive at it was to manually summarize and then evaluate sentences for common traits, which would take a lot of time, but using Machine learning we can develop an algorithm that can process text and generate summary from it so that we can use it according to our need.
Textual information in the form of digital documents quickly accumulates to huge amounts of data. Most of this large volume of documents is unstructured: it is unrestricted and has not been organized into traditional databases. Processing documents is therefore a perfunctory task, mostly due to the lack of standards. We cannot possibly create summaries of all of the text manually; there is a great need for

automatic methods and Text Summarizer helps us to achieve that summary as result of its output to a given input as text to be processed.

1.4 Type of Summarization

1. Extractive summarization
2. Abstractive summarization

Extractive Summariztion :
The extractive text summarization technique involves pulling keyphrases from the source document and combining them to make a summary. The extraction is made according to the defined metric without making any changes to the texts.Copying parts/sentences of the source text and then combines those part/sentences together to render a summary. Importance of sentence is based on linguistic and statistical features.

Abstractive Summarization :
These methods try to first understand the text and then rephrase it in a shorter manner, using possibly different words. For perfect abstractive summary, the models has to first truly understand the document and then try to express that understanding in short possibly using new words and phrases, and are much harder than extractive. It have complex capabilities like generalization, paraphrasing and incorporating real-world knowledge. Abstractive methods select words based on semantic understanding, even those words did not appear in the source documents. It aims at producing important material in a new way. They interpret and examine the text using advanced natural language techniques in order to generate a new shorter text that conveys the most critical information from the original text.Abstractive summarization systems generate new phrases, possibly rephrasing or using words that were not in the original text. Naturally abstractive approaches are harder. For perfect abstractive summary, the model has to first truly understand the document and then try to express that understanding in short possibly using new words and phrases. Much harder than extractive. Has complex capabilities like generalization, paraphrasing and incorporating real-world knowledge.

CHAPTER 2 – TECHNOLOGY USED

2.1 Machine Learning

Machine Learning is an application of artificial intelligence that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine Learning focuses on the development of computer programs that can access data and use it learn for themselves. The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.Machine Learning is the field of study that gives computers the capability to learn without being explicitly programmed. ML is one of the most exciting technologies that one would have ever come across. As it is evident from the name, it gives the computer that which makes it more similar to humans: The ability to learn. Machine learning is actively being used today, perhaps in many more places than one would expect.

Machine learning (ML) is a category of algorithm that allows software applications to become more accurate in predicting outcomes without being explicitly programmed. The basic premise of machine learning is to build algorithms that can receive input data and use statistical analysis to predict an output while updating outputs as new data becomes available.The processes involved in machine learning are similar to that of data mining and predictive modeling. Both require searching through data to look for patterns and adjusting program actions accordingly. Many people are familiar with machine learning from shopping on the internet and being served ads related to their purchase. This happens because recommendation engines use machine learning to personalize online ad delivery in almost real time. Beyond personalized marketing, other common machine learning use cases include fraud detection, spam filtering, network security threat detection, predictive maintenance and building news feeds.

2.2 Types of Machine Learning

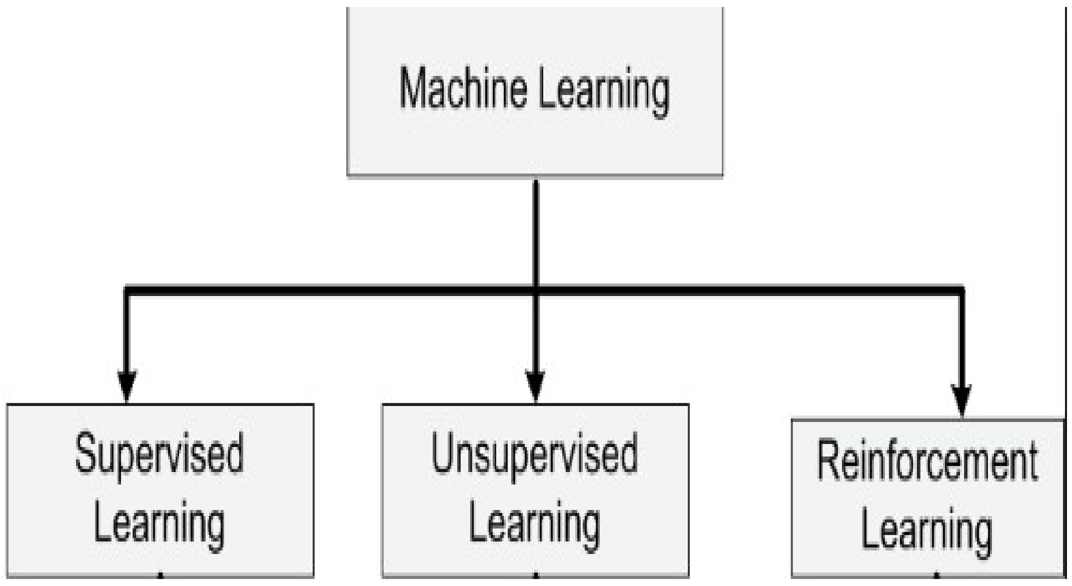


Fig 2.1 Types of Machine Learning

- **Supervised Machine Learning :** In supervised machine learning algorithms can apply what has been learned in the past to new data using labelled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.The majority of practical machine learning uses supervised learning.Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.Y = f(X) .The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers, the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

- **Unsupervised machine learning :** In Unsupervised machine learning algorithms are used when the information used to train is neither classified nor labelled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabelled data. The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabelled data.Unsupervised learning is where you only have input data (X) and no corresponding output variables.The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data.These are called unsupervised learning because unlike supervised learning above there is no correct answers and there is no teacher. Algorithms are left to their own devices to discover and present the interesting structure in the data.Unsupervised learning problems can be further grouped into clustering and association problems.Unsupervised learning is a branch of machine learning that learns from test data that has not been labeled, classified or categorized. Instead of responding to feedback, unsupervised learning identifies commonalities in the data and reacts based on the presence or absence of such commonalities in each new piece of data. Alternatives include supervised learning and reinforcement learning.Unsupervised learning problems can be further grouped into clustering and association problems. Clustering: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior. Association: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.
- **Reinforcement machine learning :** Reinforcement machine learning is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows

machines and software agents to automatically determine the ideal behaviour within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.Reinforcement is a class of machine learning where an agent learns how to behave in the environment by performing actions and thereby drawing intuitions and seeing the results.

2.3 Neural Network

A Neural Network is an information processing paradigm that is inspired by the way biological nervous system, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. The neural network itself isn't and algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs.

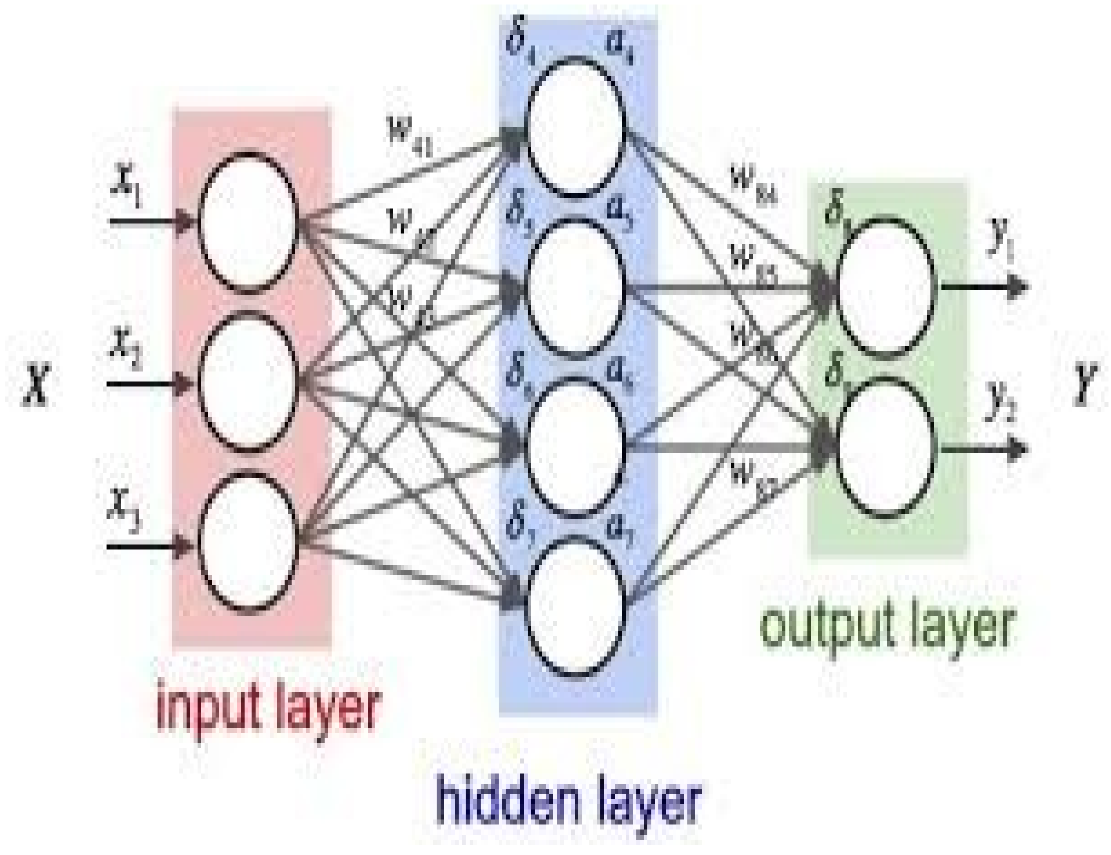


Fig 2.2 Neural Network

There are 3 parts that make up the architecture of a basic Neural Network. These are:

- Unit/Neurons
- Connections / Weights /Parameters.
- Biases.

Units / Neurons : Being the least important out of the three parts of an NNs architectures, these are functions which contain weights and biases in them and wait

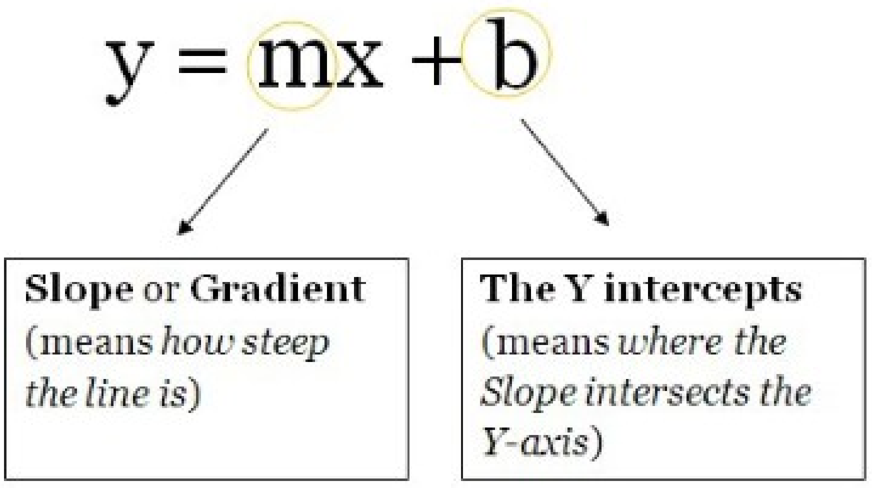

y = mx + b
Slope or Gradient
(means how steep the line is)
The Y intercepts
(means where the Slope intersects the Y-axis)

Fig 2.3 Straight Line.

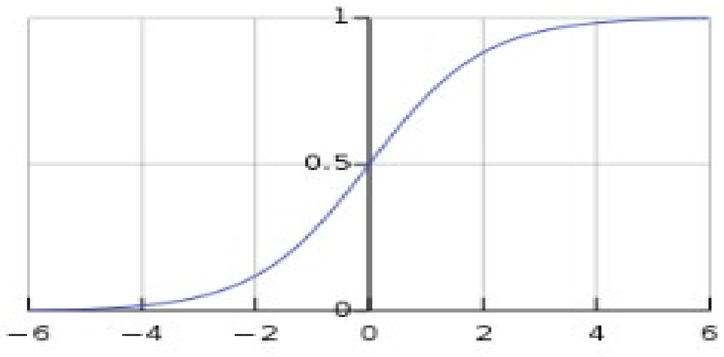
for the data to come them. After the data arrives, they, perform some computations and then use an activation function to restrict the data to a range(mostly).Think of these units as a box containing the weights and the biases. The box is open from 2 ends. One end receives data, the other end outputs the modified data. The data then starts to come into the box, the box then multiplies the weights with the data and then adds a bias to the multiplied data. This is a single unit which can also be thought of as a function. This function is similar to this, which is the function template for a straight line .

Weights / Parameters / Connections : Being the most important part of an NN, these(and the biases) are the numbers the NN has to learn in order to generalize to a problem. These are the values that are updated and responsible to get final output at output node. Weight refers to the strength or amplitude of a connection between two nodes, corresponding in biology to the amount of influence the firing of one neuron has on another. The term is typically used in artificial and biological neural network research.

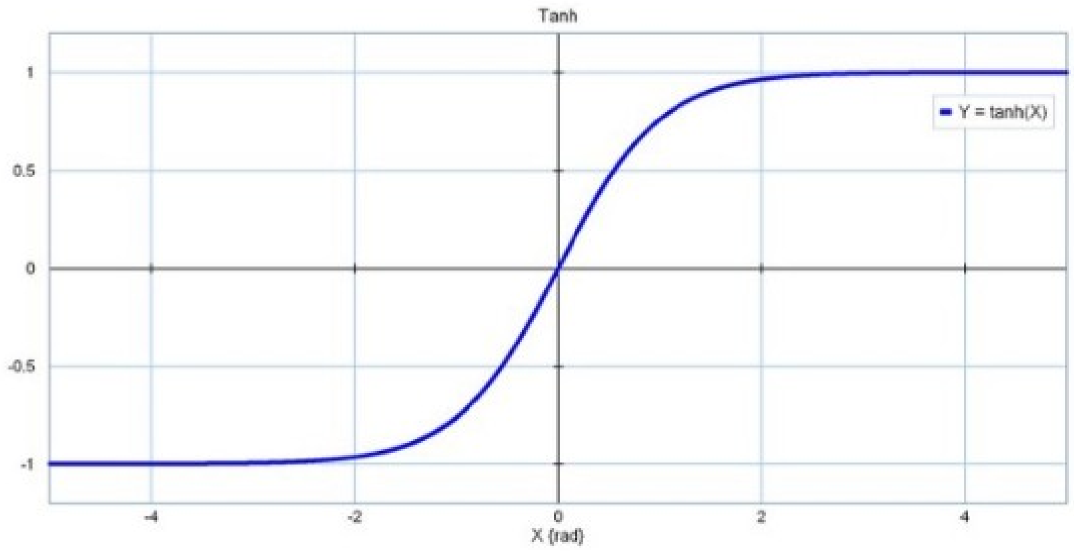
Biases : In Neural network, some inputs are provided to an artificial neuron, and with each input a weight is associated. Weight increases the steepness of activation function. This means weight decide how fast the activation function will trigger whereas bias is used to delay the triggering of the activation function.

Activation Functions :

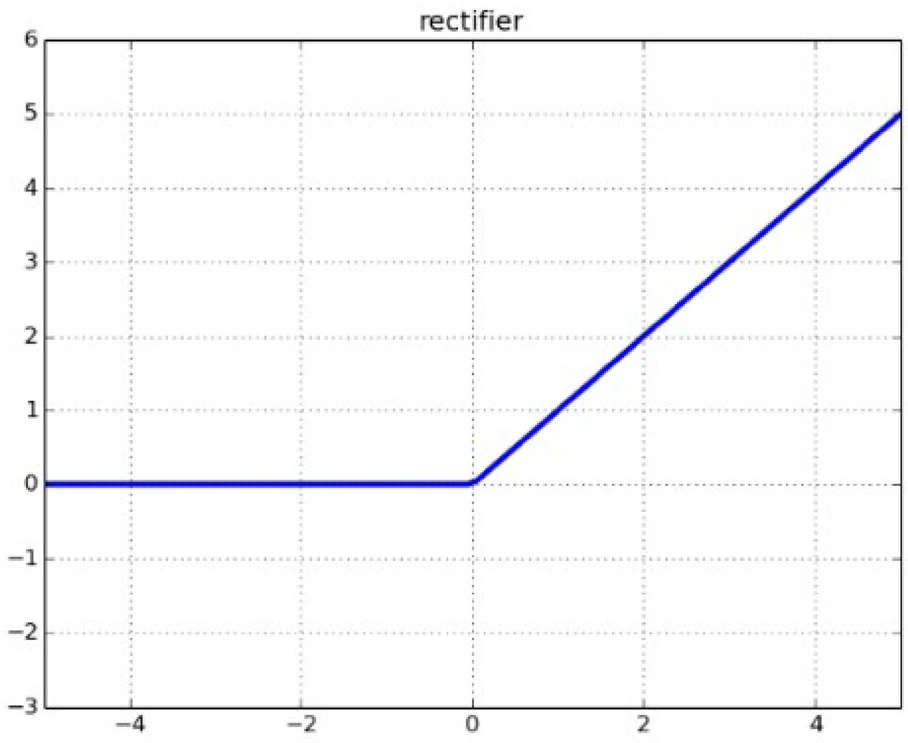
These are also known as mapping functions. They take some input on the x-axis and output a value in a restricted range(mostly). They are used to convert large outputs from the units into a smaller value—most of the times—and promote non-linearity in your Neural Network. Our choice of an activation function can drastically improve or hinder the performance of our Neural Network. we can choose different activation functions for different units if we like.Here are some activation functions :



The Sigmoid function



The tanh function



The ReLU function

Fig 2.4 Activation functions

Layers : These are what help an NN gain complexity in any problem. Increasing layers(with units) can increase the non-linearity of the output of an NN.Each layer contains some amount of Units. The amount in most cases is entirely up to the creator. However, having too many layers for a simple task can unnecessarily increase its complexity and in most cases decrease its accuracy. The opposite also holds true.

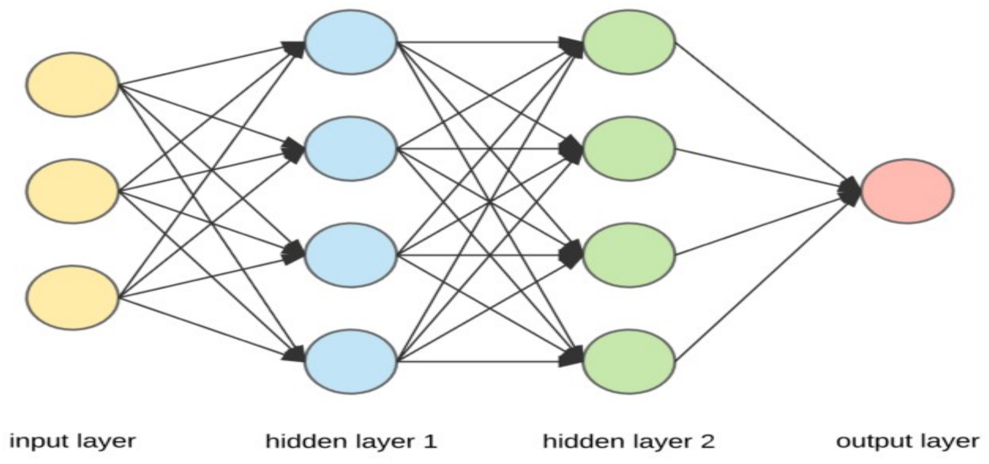
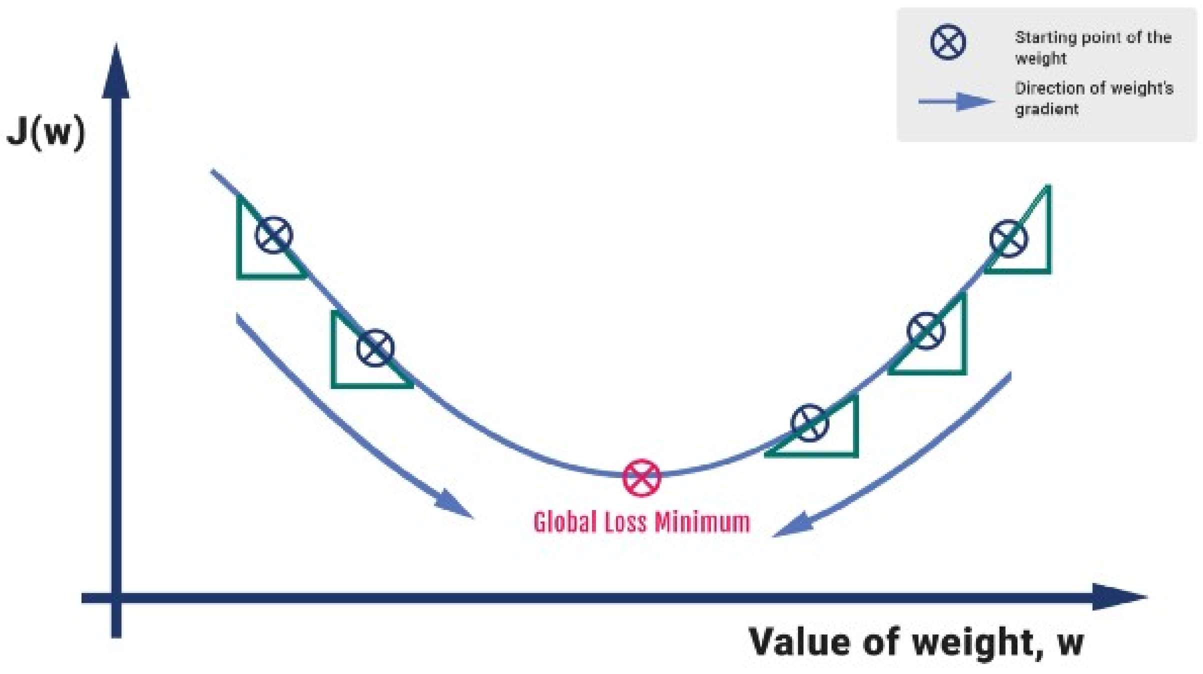


Fig 2.5 Layers

Gradient Descent : Gradient Descent is a process that occurs in the backpropagation phase where the goal is to continuously resample the gradient of the model's parameter in the opposite direction based on the weight *w*, updating consistently until we reach the global minimum of function *J(w)*.

Fig 2.6 Gradient Descent

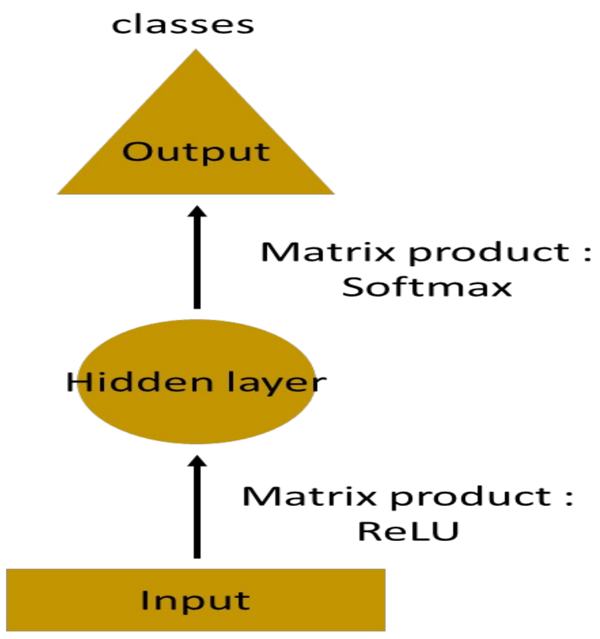


We compute the activation for the incoming parameters, we carry out feedforward by taking the weighted sum of the activation and its bias. We extract the error term of the output sample by subtracting it with the actual ‘target’ value.The gradient descent process is exhibited in the form of the backpropagation step where we compute the error vectors δ backward, starting from the final layer. Depending upon the activation function, we identify how much change is required by much change is required by taking the partial derivative of the function with respect to w.The change value gets multiplied by the learning rate. As part of the output, we subtract this value from the previous output to get the updated value. We continue this till we reach convergence.

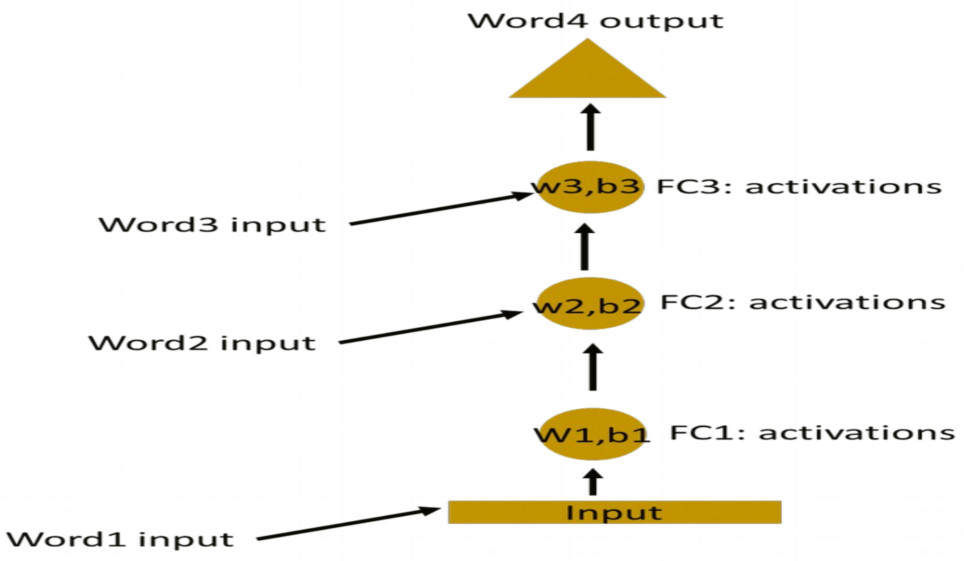
2.4 Recurrent Neural Network

A Recurrent Neural Network is a class of artificial neural network where connections between nodes form a directed graph along a sequence. This allows it to exhibit temporal dynamic behaviour for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.Recurrent Neural Networks are the state of the art algorithm for sequential data and among others used by Apples Siri and Googles Voice Search. This is because it is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for Machine Learning problems that involve sequential data. It is one of the algorithms behind the scenes of the amazing achievements of Deep Learning in the past few years.

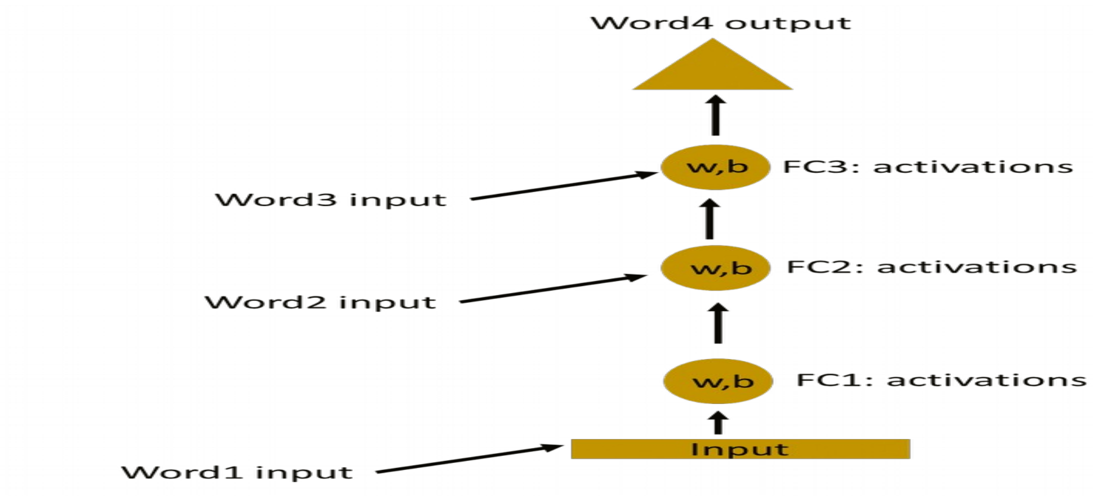
Recurrent Neural Networks (RNN) are a powerful and robust type of neural networks and belong to the most promising algorithms out there at the moment because they are the only ones with an internal memory.For example: Let’s say task is to predict the next word in a sentence. Using Neural Network in simplest form, we have an input layer, a hidden layer and an output layer. The input layer receives the input, the hidden layer activations are applied and then we finally receive the output.



Let’s have a deeper network, where multiple hidden layers are present. So here, the input layer receives the input, the first hidden layer activations are applied and then these activations are sent to next hidden layer, and successive activations through the layers to produce the output. Each hidden layer is characterized by its own weights and biases. Since each hidden layer has its own weights and activations, they behave independently. Now the objective is to identify the relationship between successive inputs.



Here, the weights and bias of these hidden layers are different. And hence each of these layers behaves independently and cannot be combined together. To combine these hidden layers together, we shall have the same weights and bias for these hidden layers.



We can now combines these layers together, that the weights and bias of all hidden layers is the same. All these hidden layers can be rolled in together in a single recurrent layer.

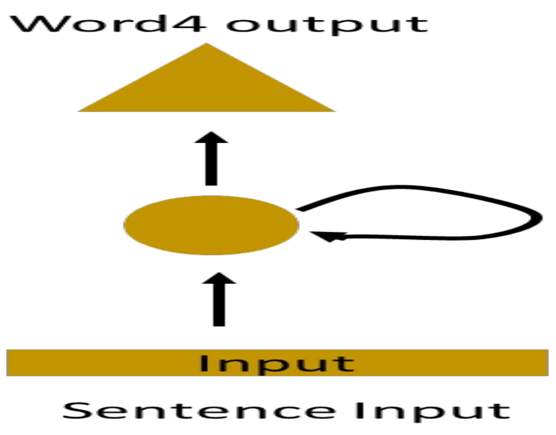


Fig 2.7 Recurrent Neural Network

So it's like supplying the input to the hidden layer. At all the time steps weights of the recurrent neuron would be the same since it's a single neuron now. So a recurrent neuron stores the state of a previous input and combines with the current input thereby preserving some relationship of the current input with the previous input.

Recurrent Neural Networks add the immediate past to the present.Therefore a Recurrent Neural Network has two inputs, the present and the recent past. This is important because the sequence of data contains crucial information about what is coming next, which is why a RNN can do things other algorithms can't. A Feed-Forward Neural Network assigns, like all other Deep Learning algorithms, a weight matrix to its inputs and then produces the output. Note that RNN's apply weights to the current and also to the previous input. Furthermore they also tweak their weights for both through gradient descent and Backpropagation Through Time.

We can view a RNN as a sequence of Neural Networks that you train one after another with backpropagation. The image below illustrates an unrolled RNN. On the left, you can see the RNN, which is unrolled after the equal sign. Note that there is no cycle after the equal sign since the different timesteps are visualized and information gets's passed from one timestep to the next. This illustration also shows why a RNN can be seen as a sequence of Neural Networks. If we do Backpropagation Through Time, it is required to do the conceptualization of unrolling, since the error of a given timestep depends on the previous timestep.

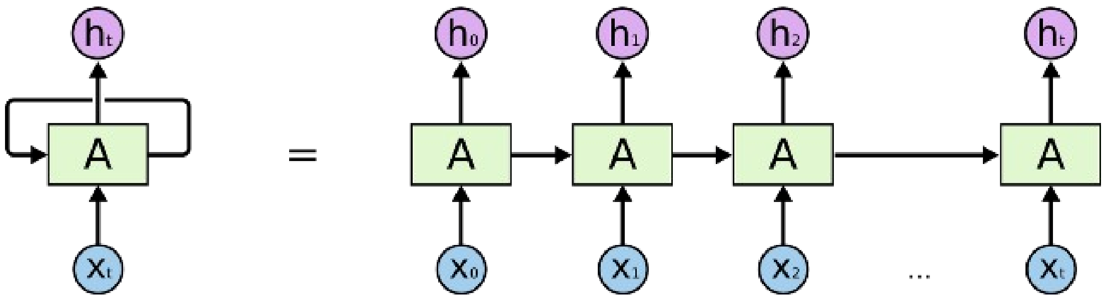


Fig 2.8 Unrolled Recurrent Neural Network.

Issues in RNN'S: There are two major obstacles RNN's have or had to deal with. But to understand them, We first need to know what a gradient is.A gradient is a partial derivative with respect to its inputs. If you don't know what that means, just think of it like this: A gradient measures how much the output of a function changes, if We change the inputs a little bit. We can also think of a gradient as the slope of a function. The higher the gradient, the steeper the slope and the faster a model can learn. But if the slope is zero, the model stops to learning. A gradient simply measures the change in all weights with regard to the change in error.

- **Exploding Gradients :**“Exploding Gradients“ occurs when the algorithm assigns a stupidly high importance to the weights, without much reason. But fortunately, this problem can be easily solved if we truncate or squash the gradients.
- **Vanishing Gradients : “**Vanishing Gradients“ when the values of a gradient are too small and the model stops learning or takes way too long because of that.

2.5 LSTM

Long Short-Term Memory (LSTM) networks are an extension for recurrent neural networks, which basically extends their memory. Therefore it is well suited to learn from important experiences that have very long time lags in between.The LSTM is RNN architecture which can remember past contextual values. These stored values do not change over time while training the model. There are four components in LSTM which are LSTM Units, LSTM Blocks, LSTM Gates and LSTM Recurrent

Components. LSTM Unit store values for long time or for short time. LSTM has no activation functions for their recurrent components. Since there are no activation function the values of units does not change for some period until the context is changed. A LSTM Block contains such many units. LSTM’s are considered as deep neural networks. These LSTM’s are implemented in parallel systems.

LSTM blocks have four gates to control the information flow. Logistic functions are used to implement these gates, to compute a value between 0 and 1. To allow or deny information flow into or out of the memory, multiplication of values with these logistic functions are done. To control the flow of new values into memory, input gate plays key role. The extent to which a value remains in memory is controlled by forget gate. Output gate controls the extent to which the value in memory is used to compute the output activation of the block. Sometimes in implementations, the input and forget gates are merged into a single gate, hence we can see even 3 gate representations of LSTM. When new value which is worth remembering is available then we can forget the old value. This represents the combining effect of input and forget gate of LSTM.

LSTM’s enable RNN’s to remember their inputs over a long period of time. This is because LSTM’s contain their information in a memory, that is much like the memory of a computer because the LSTM can read, write and delete information from its memory. This memory can be seen as a gated cell, where gated means that the cell decides whether or not to store or delete information (e.g if it opens the gates or not), based on the importance it assigns to the information. The assigning of importance happens through weights, which are also learned by the algorithm. This simply means that it learns over time which information is important and which not.

In an LSTM you have three gates: input, forget and output gate. These gates determine whether or not to let new input in (input gate), delete the information because it isn’t important (forget gate) or to let it impact the output at the current time step (output gate). We can see an illustration of a RNN with its three gates below. The gates in a LSTM are analog, in the form of sigmoids, meaning that they range from 0 to 1. The fact that they are analog, enables them to do backpropagation with it.

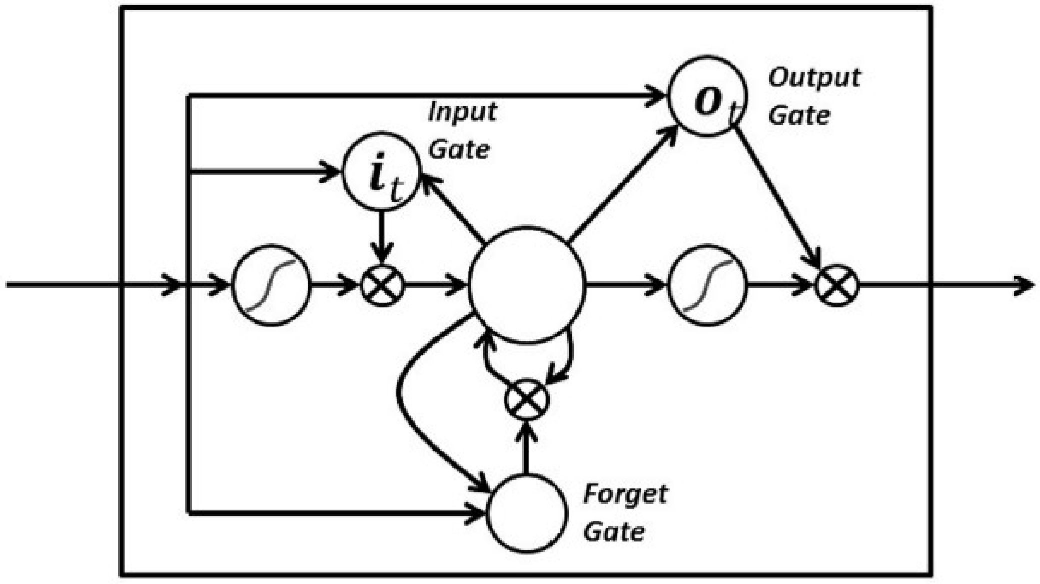


Fig 2.9 LSTM Cell

2.6 Encoder and Decoders

One approach to seq2seq prediction problems that has proven very effective is called the Encoder-Decoder LSTM. This architecture is comprised of two models: one for reading the input sequence and encoding it into a fixed-length vector, and a second for decoding the fixed-length vector and outputting the predicted sequence. The use of the models in concert gives the architecture its name of Encoder-Decoder LSTM designed specifically for seq2seq problems.

The Encoder-Decoder LSTM was developed for natural language processing problems where it demonstrated state-of-the-art performance, specifically in text translation called statistical machine translation. The innovation of this architecture is the use of a fixed-sized internal representation in the heart of the model that input sequences are read to and output sequences are read from. For this reason, the method may be referred to as sequence embedding. In one of the first applications of the architecture to English-to-French translation, the internal representation of the encoded English phrases was visualized. The plots revealed a qualitatively meaningful learned structure of the phrases harnessed for the translation task. On the task of translation, the model was found to be more effective when the input sequence was reversed. Further, the model was shown to be effective even on very long input sequences. This approach has also been used with image inputs where a Convolutional Neural Network is used as a feature extractor on input images, which is then read by a decoder LSTM.

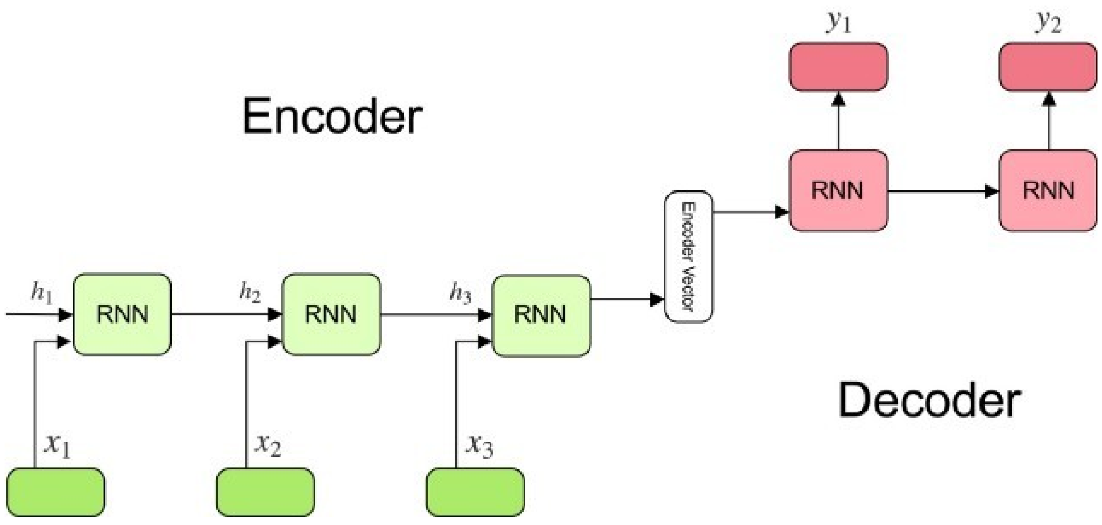


Fig 2.10 Encoder and Decoder

Encoder : A stack of several recurrent units (LSTM cells for better performance) where each accepts a single element of the input sequence, collects information for that element and propagates it forward. Each word is represented as *x_i* where *i* is the order of that word. The hidden states h_i are computed using the formula:

$$h_t = f(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$$

This simple formula represents the result of an ordinary recurrent neural network. We just apply the appropriate weights to the previous hidden state *h_(t-1)* and the input vector *x_t.* Encoder Vector is the final hidden state produced from the encoder part of the model. It is calculated using the formula above. This vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions.It acts as the initial hidden state of the decoder part of the model.

Decoder : A stack of several recurrent units where each predicts an output y_t at a time step t. Each recurrent unit accepts a hidden state from the previous unit and produces and output as well as its own hidden state. Any hidden state h_i is computed using the formula:

$$h_t = f(W^{(hh)}h_{t-1})$$

The output y_t at time step t is computed using the formula:

2.7 Word Embedding

Before diving deep in what are word embeddings? Why it is used? Etc. We will see some examples first:

1. There are many websites that ask us to give reviews or feedback about their product when we are using them. like: - Amazon, IMDB.

2. we also use to search at google with couple of words and get result related to it.

3. There are some sites that put tags on the blog related the material in the blog.

So how these all are done. These things are application of Text processing. we use text to do sentiment analysis, clustering similar word, document classification and tagging. As we read any newspaper we can say that what is the news about but how computer will do these things? Computer can match strings and can tell us that they are same or not but how do we make computers tell you about football or Ronaldo when you search for Messi?.

For tasks like object or speech recognition we know that all the information required to successfully perform the task is encoded in the data (because humans can perform these tasks from the raw data). However, natural language processing systems traditionally treat words as discrete atomic symbols, and therefore 'cat' may be represented as Id537 and 'dog' as Id143. These encodings are arbitrary, and provide no useful information to the system regarding the relationships that may exist between the individual symbols.

There are two basic types of Word Embeddings:

1. Frequency Based Embedding
    a. Count Vector
    b. TF-IDF Vectorization
    c. Co-Occurrence Matrix with a fixed context window
2. Prediction Based Embeddings
    a. Continuous Bag of words (CBOW) Abstractive Text Summarizer
    b. Skip-gram Model

In our project we are using ConceptNet Numberbatch word embedding. ConceptNet Numberbatch consists of state-of-the-art semantic vectors (also known as word embeddings) that can be used directly as a representation of word meanings or as a starting point for further machine learning. ConceptNet Numberbatch is part of the ConceptNet open data project. ConceptNet provides lots of ways to compute with word meanings, one of which is word embeddings. ConceptNet Numberbatch is a snapshot of just the word embeddings.

CHAPTER 3 -HARDWARE AND SOFTWARE REQUIREMENTS.

3.1 Hardware Requirements

- Processor – Intel i5.
- Hard Disk – 1 TB.
- Memory – 8 GB.
- Graphic Memory – 2 GB (not necessary)

3.2 Software Requirement

3.2.1 Python 3.6.7

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms. The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, http://www.python.org/, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation. The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications. This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

3.2.2 Visual Studio code

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity)

3.3 Libraries

3.3.1 Pandas :

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal.

3.3.2 Numpy:

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multidimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. NumPy is licensed under the BSD license, enabling reuse with few restrictions.

3.3.3 re

This module provides regular expression matching operations similar to those found in Perl. Both patterns and strings to be searched can be Unicode strings as well as 8-bit strings.

Regular expressions use the backslash character ('\’) to indicate special forms or to allow special characters to be used without invoking their special meaning. This collides with Python’s usage of the same character for the same purpose in string literals; for example, to match a literal backslash, one might have to write '\\\\' as the pattern string, because the regular expression must be \\, and each backslash must be expressed as \\ inside a regular Python string literal.

3.3.4 NLTK

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

Example :

```
>>> import nltk

>>> sentence = """At eight o'clock on Thursday morning
... Arthur didn't feel very good."""

>>> tokens = nltk.word_tokenize(sentence)

>>> tokens

['At', 'eight', "o'clock", 'on', 'Thursday', 'morning',
'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']
```

We have used nltk stopword in project.Stopwords are the English words which does not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence. For example, the words like the, he, have etc. Such words are already captured this in corpus named corpus.

3.3.5 Tensorflow

TensorFlow is a computational framework for building machine learning models. It is the second generation system from Google Brain headed by Jeff Dean. Launched in early 2017, it has disrupted the ML world by bringing in numerous capabilities from scalability to building production ready models.

TensorFlow provides a variety of different tool kits that allow you to write code at your preferred level of abstraction. For instance, you can write code in the Core TensorFlow (C++) and call that method from Python code. You can also define the architecture on which your code should run (CPU, GPU etc.). In the above hierarchy, the lowest level in which you can write your code is C++ or Python. These two levels allow you to write numerical programs to solve mathematical operations and equations. Although this is not highly recommended for building Machine Learning models, it offers a wide range of math libraries that ease your tasks. The next level in which you can write your code is using the TF specific abstract methods which are highly optimised for model components. For example, using the tf.layers method abstract you can play with the layers of a neural net. We can build a model and evaluate the model performance using the tf.metrics method.

A session encapsulates the state of the TensorFlow runtime, and runs TensorFlow operations. Every line of code you write using TensorFlow is represented by an underlying graph.The idea behind utilising graphs is to create portable code. Yes, this graph can be exported and used by anybody on any type of architecture. But, why does TensorFlow not compute the results? Because, it follows the lazy evaluation paradigm. All graphs created are tied to a session and we have to tell TensorFlow to compute the results using session.run. For example: (tensorflow as tf)

```
session = tf.Session()

x = tf.constant([1, 2, 3], name = "x")

y = tf.constant([4, 5, 6], name = "y")

z2 = x * y

session.run(z3)
```

3.4 Setting up Environment for Summarizer Using Python

Python is a high-level, interpreted programming language, created by Guido van Rossum. The language is very popular for its code readability and compact line of codes. It uses white space inundation to delimit blocks. Python provides a large standard library which can be used for various applications for example natural language processing, machine learning, data analysis etc. It is favoured for complex projects, because of its simplicity, diverse range of features and its dynamic nature. The following components are required to be downloaded and installed properly.

- Download and install python version == 3.6 or above for https://www.python.org/downloads/
- Download and install visual studio code form https://code.visualstudio.com/download
- install following package using command pip install package_name
    - Numpy
    - Pandas
    - Nltk
    - Tensorflow
- Download dataset amazon-fine-food-reviews from https://www.kaggle.com/snap/amazon-fine-food-reviews/version/2?

CHAPTER 4 – WORKING

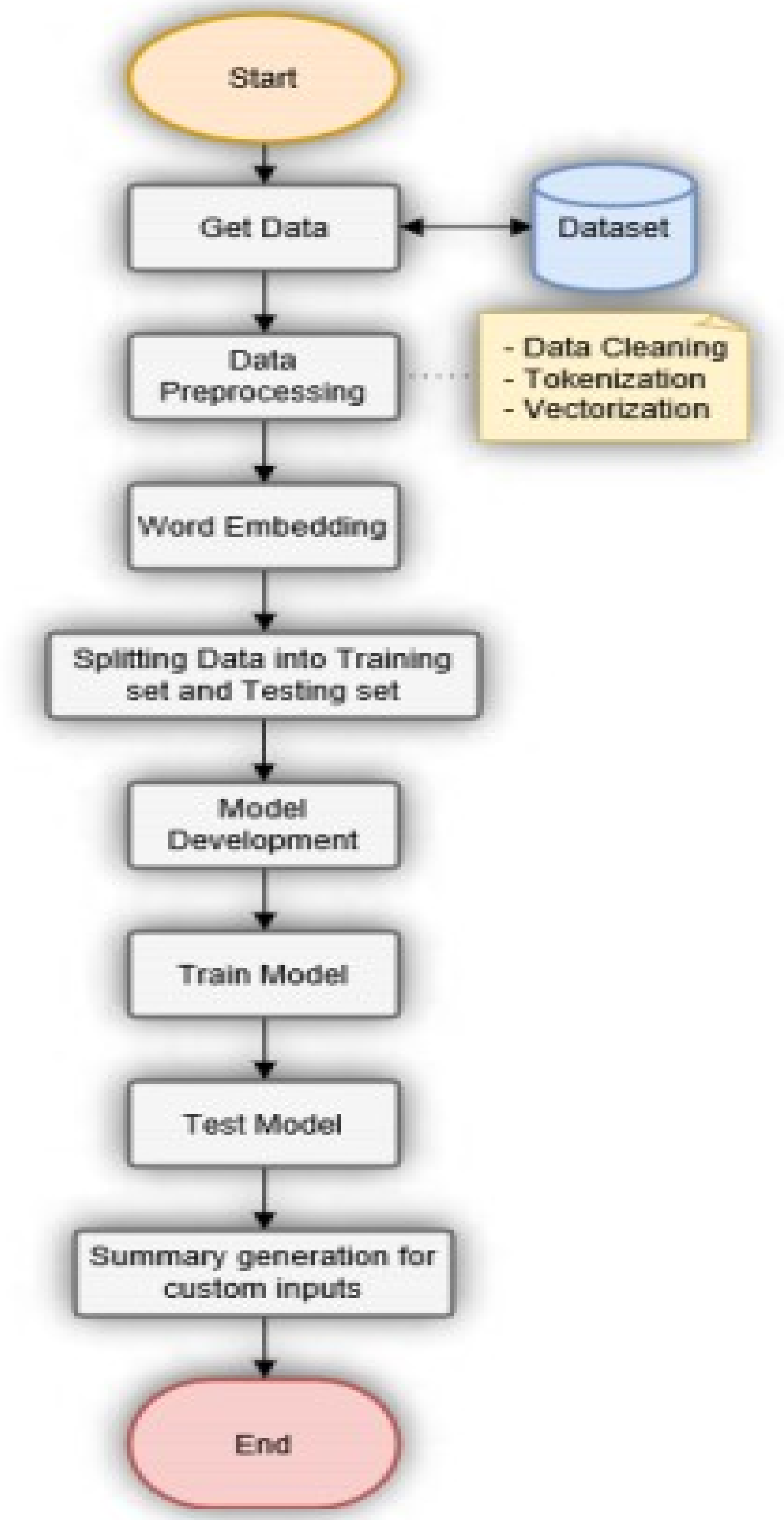
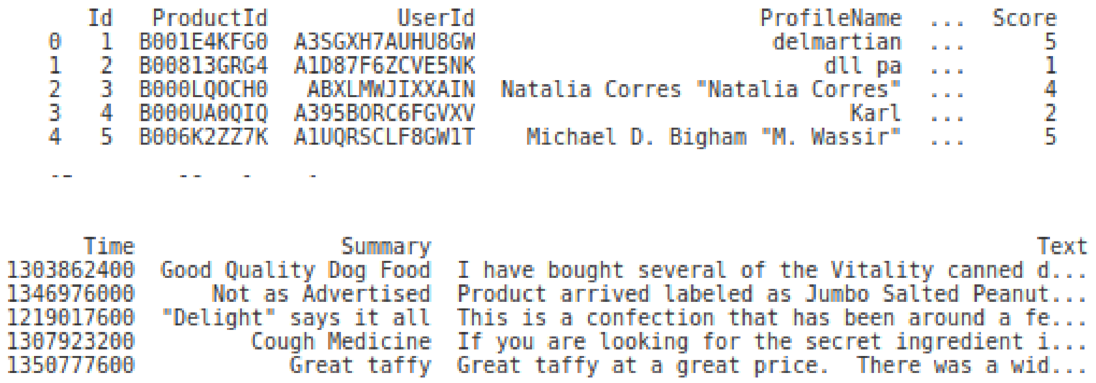


Fig 4.1 Project Approach

4.1 Data Processing and model development

- The above figure 4. shows the flow chart of the project approach. The first step is to acquire data, which we did in setting up section. The data consists of 10 columns and more than 500000 rows. The data includes products and user information, ratings, and a plain text review with its summary. When download the dataset, we get a CSV file. This file is then read by program and in second step it is cleaned. The preview of data can be viewed in figure 4.2.

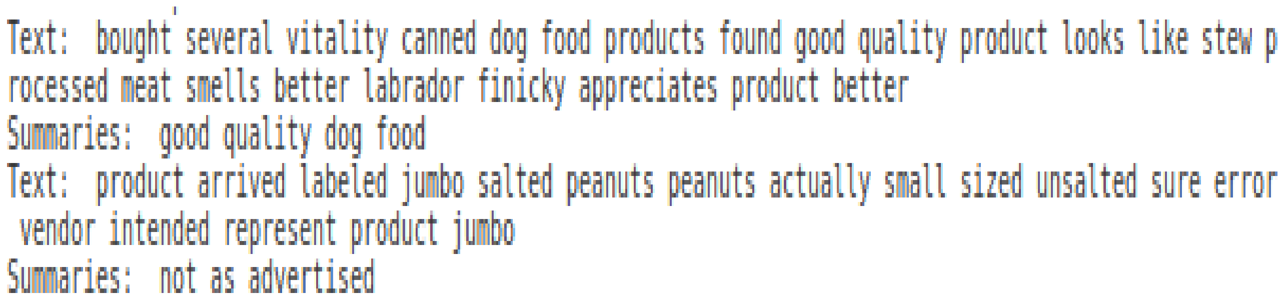


```
   Id   ProductId          UserId                      ProfileName  ...  Score
0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian  ...      5
1   2  B00813GRG4  A1D87F6ZCVE5NK                           dll pa  ...      1
2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"  ...      4
3   4  B000UA0QIQ  A395BORC6FGVXV                             Karl  ...      2
4   5  B006K2ZZ7K  A1UQRSCLF8GW1T    Michael D. Bigham "M. Wassir"  ...      5


      Time                Summary                                               Text
1303862400  Good Quality Dog Food  I have bought several of the Vitality canned d...
1346976000      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
1219017600  "Delight" says it all  This is a confection that has been around a fe...
1307923200         Cough Medicine  If you are looking for the secret ingredient i...
1350777600            Great taffy  Great taffy at a great price.  There was a wid...
```

Fig 4.2 amazon fine food review dataset

- The cleaning phase consists of removing stopwords, removing null entries, replacing contractions with their longer forms, and removing unwanted characters like symbols and emojis. Stopwords are only removed from Text part of the reviews, they are not removed from Summaries to sound like more natural phrases. After processing the data, it will have only two columns i.e. text and summary. Figure 9 shows the cleaned data.



```
Text:  bought several vitality canned dog food products found good quality product looks like stew p
rocessed meat smells better labrador finicky appreciates product better
Summaries:  good quality dog food
Text:  product arrived labeled jumbo salted peanuts peanuts actually small sized unsalted sure error
 vendor intended represent product jumbo
Summaries:  not as advertised
```

Fig 4.3 clean dataset

- Once the data is cleaned, it is tokenized and converted into vectors so that it can be processed by the model. We then load our pre-trained ConceptNet Numberbatch word embedding. We find the number of words missing in the embedding by comparing all the tokens from our reviews data to loaded embedding. We also add <unk> token for those which words are not known and we add <pad> in the end of line to indicate that it's the end of line. Finally, we sort the summary and text by the length of text, i.e. from shortest to longest.

- Now we will design our Model which will be used while training. Since we are using TensorFlow, our major effort goes into building graphs. We will create few place holders to hold our data and other parameters. Then we will create our bi-directional encoder, which will generate its own representation of input data. The basic architecture used for this approach can be seen in figure 4.4.

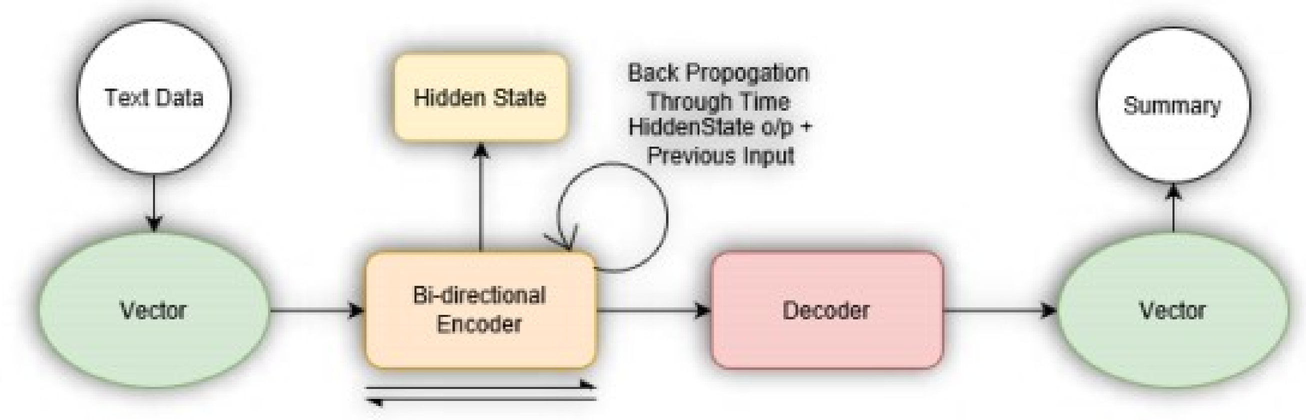


Fig 4.4 Architecture of model

- While designing Decoder we will implement Bahdanau Attention. Let x denote the input sentence consisting of a sequence of M words x = [x1, . . . ,xM], where each word xi is part of vocabulary V, of size |V| = V . Our task is to generate a target sequence y = [y1, . . . , yN], of N words, where N < M, such that the meaning of x is preserved: y = argmaxy P(y|x), where y is a random variable denoting a sequence of N words. Typically the conditional probability is modeled by a parametric function with parameters θ: P(y|x) = P(y|x; θ). Training involves finding the θ which maximizes the conditional probability of sentence-summary pairs in the training corpus. If the model is trained to generate the next word of the summary, given the previous words, then the above conditional can be factorized into a product of individual conditional probabilities:

$$P(y|x;\theta) = \Pi^{N}{}_{t=1}\, p(y_t \mid \{y_1,\dots,y_{t-1}\}, x; \theta)$$

  This Conditional probability is implemented using RNN Encoder-Decoder. This model is also called as Recurrent Attentive Summarizer

- The LSTM decoder is defined as :

$$\begin{aligned}
i_t &= \sigma(W_1 y_{t-1} + W_2 h_{t-1} + W_3 c_t) \\
i'_t &= \tanh(W_4 y_{t-1} + W_5 h_{t-1} + W_6 c_t) \\
f_t &= \sigma(W_7 y_{t-1} + W_8 h_{t-1} + W_9 c_t) \\
o_t &= \sigma(W_{10} y_{t-1} + W_{11} h_{t-1} + W_{12} c_t) \\
m_t &= m_{t-1} \odot f_t + i_t \odot i'_t \\
h_t &= m_t \odot o_t \\
P_t &= \rho(W_{13} h_t + W_{14} c_t).
\end{aligned}$$

- model input place holder

```
def model_inputs():
    #Palceholders for inputs to the model
    input_data = tf.placeholder(tf.int32, [None, None], name='input')
    targets = tf.placeholder(tf.int32, [None, None], name='targets')
    lr = tf.placeholder(tf.float32, name='learning_rate')
    keep_prob = tf.placeholder(tf.float32, name='keep_prob')
    summary_length = tf.placeholder(tf.int32, (None,), name='summary_length')
    max_summary_length = tf.reduce_max(summary_length, name='max_dec_len')
    text_length = tf.placeholder(tf.int32, (None,), name='text_length')
    return input_data, targets, lr, keep_prob, summary_length,
    max_summary_length, text_length
```

- Building the model: There are quite a few placeholders that we need to make for this model. Most of them are self explanatory, but the just to be clear on a few, summary_lengthand text_length are the lengths of each sentence within a batch, and max_summary_length is the maximum length of a summary within a batch.

- Although the decoding layer may look a bit complex, it can be broken down into three parts: decoding cell, attention, and getting our logits.Decoding Cell: Just a two layer LSTM with dropout.

  DynamicAttentionWrapper applies the attention mechanism to our decoding cell.

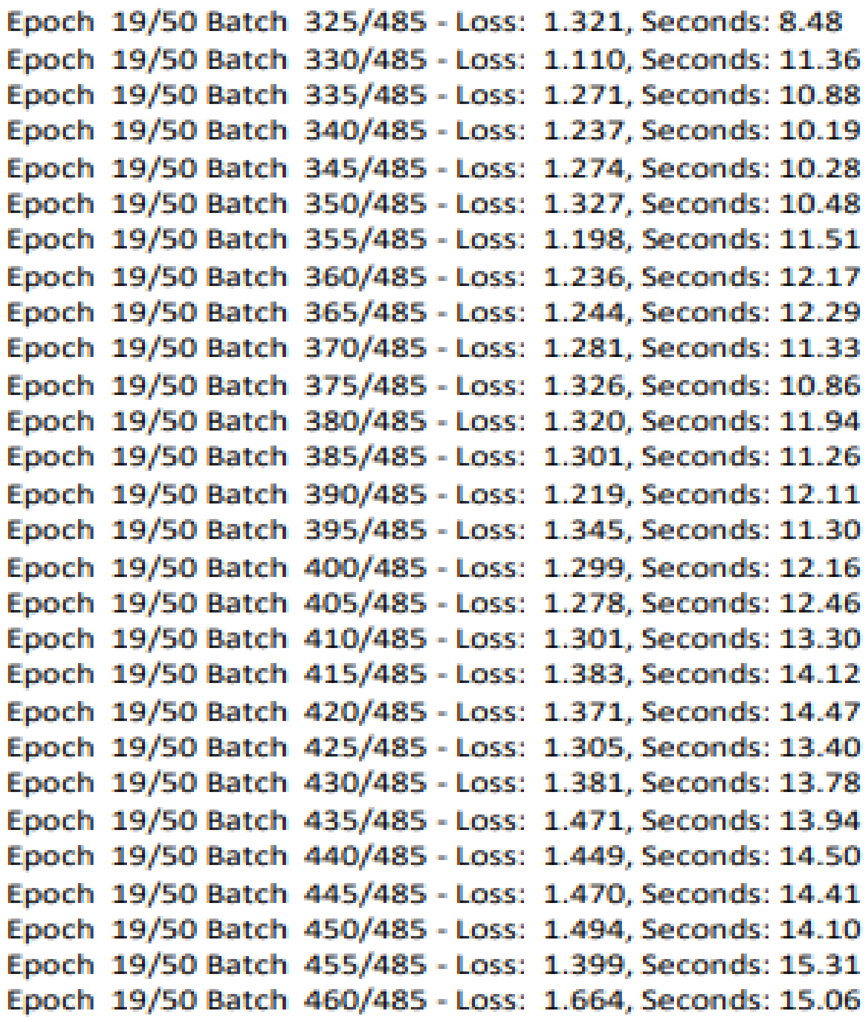
DynamicAttentionWrapperState creates the initial state that we use for our training and inference layers. Since we are using a bidirectional RNN in our decoding layer, we can only using either the forward or backward state.

- Adam is an optimization algorithm that can used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training.Adam is an adaptive learning rate optimization algorithm that's been designed specifically for training deep neural networks. First published in 2014, Adam was presented at a very prestigious conference for deep learning practitioners—ICLR 2015. The paper contained some very promising diagrams, showing huge performance gains in terms of speed of training.

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t$$
$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla w_t)^2$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$
$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} * \hat{m}_t$$

Actual step size taken by the Adam in each iteration is approximately bounded the step size hyper-parameter. This property add intuitive understanding to previous unintuitive learning rate hyper-parameter. Adam was designed to combine the advantages of Adagrad, which works well with sparse gradients, and RMSprop, which works well in on-line settings. Having both of these enables us to use Adam for broader range of tasks.

4.2 Training the model



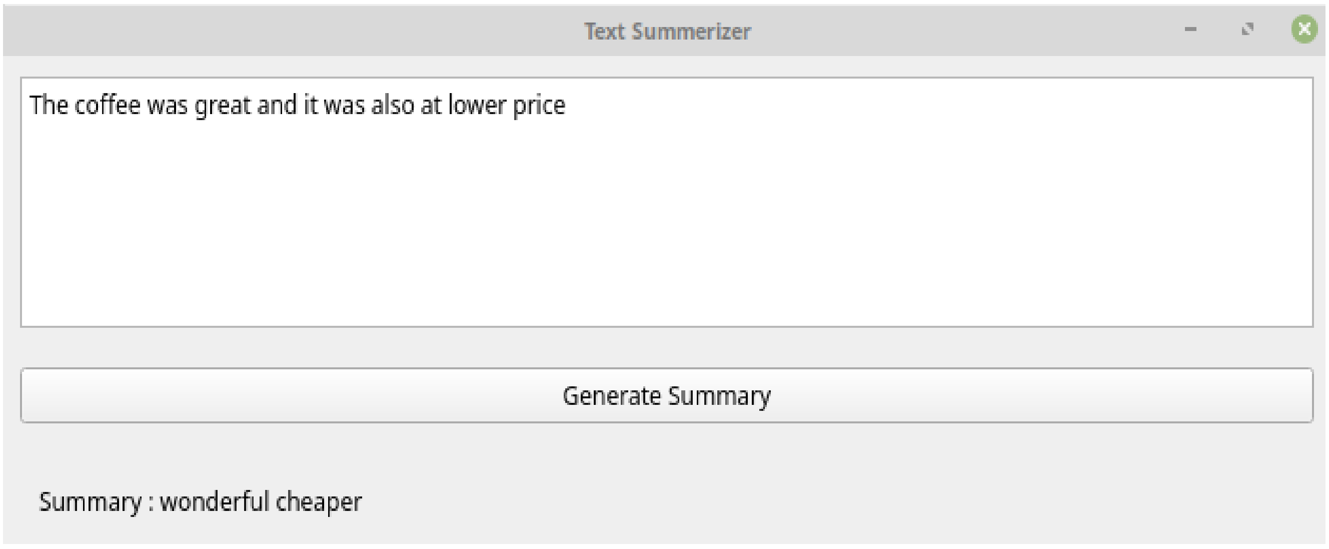
```
Epoch  19/50 Batch  325/485 - Loss:  1.321, Seconds: 8.48
Epoch  19/50 Batch  330/485 - Loss:  1.110, Seconds: 11.36
Epoch  19/50 Batch  335/485 - Loss:  1.271, Seconds: 10.88
Epoch  19/50 Batch  340/485 - Loss:  1.237, Seconds: 10.19
Epoch  19/50 Batch  345/485 - Loss:  1.274, Seconds: 10.28
Epoch  19/50 Batch  350/485 - Loss:  1.327, Seconds: 10.48
Epoch  19/50 Batch  355/485 - Loss:  1.198, Seconds: 11.51
Epoch  19/50 Batch  360/485 - Loss:  1.236, Seconds: 12.17
Epoch  19/50 Batch  365/485 - Loss:  1.244, Seconds: 12.29
Epoch  19/50 Batch  370/485 - Loss:  1.281, Seconds: 11.33
Epoch  19/50 Batch  375/485 - Loss:  1.326, Seconds: 10.86
Epoch  19/50 Batch  380/485 - Loss:  1.320, Seconds: 11.94
Epoch  19/50 Batch  385/485 - Loss:  1.301, Seconds: 11.26
Epoch  19/50 Batch  390/485 - Loss:  1.219, Seconds: 12.11
Epoch  19/50 Batch  395/485 - Loss:  1.345, Seconds: 11.30
Epoch  19/50 Batch  400/485 - Loss:  1.299, Seconds: 12.16
Epoch  19/50 Batch  405/485 - Loss:  1.278, Seconds: 12.46
Epoch  19/50 Batch  410/485 - Loss:  1.301, Seconds: 13.30
Epoch  19/50 Batch  415/485 - Loss:  1.383, Seconds: 14.12
Epoch  19/50 Batch  420/485 - Loss:  1.371, Seconds: 14.47
Epoch  19/50 Batch  425/485 - Loss:  1.305, Seconds: 13.40
Epoch  19/50 Batch  430/485 - Loss:  1.381, Seconds: 13.78
Epoch  19/50 Batch  435/485 - Loss:  1.471, Seconds: 13.94
Epoch  19/50 Batch  440/485 - Loss:  1.449, Seconds: 14.50
Epoch  19/50 Batch  445/485 - Loss:  1.470, Seconds: 14.41
Epoch  19/50 Batch  450/485 - Loss:  1.494, Seconds: 14.10
Epoch  19/50 Batch  455/485 - Loss:  1.399, Seconds: 15.31
Epoch  19/50 Batch  460/485 - Loss:  1.664, Seconds: 15.06
```

Fig 4.5 Training model

Training model means reducing loss .In terms of loss function, we want our loss function to much lower in the end of training. Improving the network is possible, because we can change its function by adjusting weights. We want to find another function that performs better than the initial one.There are a lot of algorithms that optimize functions. These algorithms can gradient-based or not, in sense that they are not only using the information provided by the function, but also by its gradient. One of the simplest gradient-based algorithms. We have use adam optimizer which is modified version of gradient descent algorithm to train model faster.

4.3 Making your own summaries

```
loaded_graph = tf.Graph()
with tf.Session(graph=loaded_graph) as sess:
    loader = tf.train.import_meta_graph(checkpoint + '.meta')
    loader.restore(sess, checkpoint)
    input_data = loaded_graph.get_tensor_by_name('input:0')
    logits = loaded_graph.get_tensor_by_name('predictions:0')
    text_length = loaded_graph.get_tensor_by_name('text_length:0')
    summary_length = loaded_graph.get_tensor_by_name('summary_length:0')
    keep_prob = loaded_graph.get_tensor_by_name('keep_prob:0')
    for i, text in enumerate(texts):
        generagte_summary_length = generagte_summary_length_list[i]
                answer_logits = sess.run(logits, {input_data: [text]*batch_size,
                summary_length: [generagte_summary_length], #summary_length:
                [np.random.randint(5,8)],
                text_length: [len(text)]*batch_size,
                keep_prob: 1.0})[0]
        # Remove the padding from the summaries
        pad = vocab_to_int["<PAD>"]
        print('- Review:\n\r {}'.format(input_sentences[i]))
        l=len(int_to_vocab)
        print('- Summary:\n\r {}\n\r\n\r'.format(" ".join([int_to_vocab[j] for j in
                              answer_logits if (j!=pad and j<l) ])))
```



The coffee was great and it was also at lower price

Summary : wonderful cheaper

Fig 4.6 Generating Summary

CHAPTER -5 CONCLUSION

People need to learn much from texts, but they tend to want to spend less time while doing this. It aims to solve this problem by supplying them the summaries of the text from which they want to gain information. Goals of this project are that these summaries will be as important as possible in the aspect of the texts intention. The user will be eligible to select the summary length. Supplying the user, a smooth and clear interface to use the facility in order to latest technology and applying it for saving time and increasing efficiency and productivity of their work.

Machine Learning provides us important technique to handle situation which are related to human behaviour and used on daily basis that allow us to save time and money and focus on important goal. Text summarizer provides users to help use of machine learning algorithm and to use for their benefit as reading is an important part of life. And to read less and gain more information from the document and text we have to handle in our life we need to be faster and focus on important piece of information rather than wasting our whole time on reading unnecessary text and to avoid this text summarizer provides us feature that we can use to apply in real life.

We have successfully implemented state-of-the-art model for abstractive sentence summarization to a recurrent neural network architecture. The model is a simplified version of the encoder-decoder framework for machine translation . The model is trained on the Amazon-fine-food-review corpus to generate summaries of review based on the first line of each review. There are few limitations of the model which can be improved in further work. First limitation is that it sometimes generates repeated words in the summary, the other problem is it takes too much time to generate a summary if the input text size is large enough, the other issue is that for large text input it sometimes miss interpret the context and generates exactly opposite context summary.

CHAPTER 6 – REFERENCES

- Deep Learning for Computer Vision by Rajalingappaa Shanmugamani
- Machine Learning Algorithms by Giuseppe Bonaccorso
- Thoughtful Machine Learning with Python by Matthew Kirk
- https://towardsdatascience.com/text-summarization-with-amazon-reviews-41801c2210b
- https://towardsdatascience.com/building-your-first-neural-network-using-keras-29ad67075191
- https://towardsdatascience.com/tensorflow-the-core-concepts-1776ea1732fa
- Machine Learning for Developers by Rodolfo Bonnin