**A Project Report**

**on**

# SUGGESTION MINING USING ATTENTION NETWORKS

**Submitted in partial fulfillment of the requirements for the award of the degree of**

**BACHELOR OF TECHNOLOGY**

**in**

**INFORMATION TECHNOLOGY**

**by**

**E. Meghana(16WH1A1216)**

**K. Samhitha(16WH1A1229)**

**M. Haritha(16WH1A1247)**

**Under the esteemed guidance of**

**Mr. L. Naveen Kumar**
**Assistant Professor**


**Department of Information Technology**
**BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN**
(Accredited by NBA & NAAC '**A**' Grade, Approved by AICTE, and
Affiliated to JNTUH, Hyderabad)
**Bachupally, Hyderabad – 500090**

**April 2020**

# DECLARATION

We hereby declare that the work presented in this project entitled **"SUGGESTION MINING USING ATTENTION NETWORKS"**submitted towards completion of the major project in IV year of B.Tech IT at "BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN**"**, is an authentic record of our original work carried out under the esteem guidance of

Mr. L. Naveen Kumar, Assistant Professor, IT department.

**E.Meghana**
**(16WH1A1216)**

**K.Samhitha**
**(16WH1A1229)**

**M.Haritha**
**(16WH1A1247)**

# BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN

(Accredited by NBA & NAAC 'A' Grade, Approved by AICTE, and Affiliated to JNTUH, Hyderabad)

**Bachupally, Hyderabad – 500090**

**Department of Information Technology**


## Certificate

This is to certify that the Project report on "**Suggestion Mining Using Attention Networks**" is a bonafide work carried by **E.Meghana(16WH1A1216), k.Samhitha(16WH1A1229), M.Haritha(16WH1A1247)** in the partial fulfillment for the award of B.Tech degree in **Information Technology, BVRIT HYDERABAD COLLEGE OF ENGINEERING FOR WOMEN, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University, Hyderabad under my guidance and supervision.

The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma.

**Internal Guide**
**Mr. L. Naveen Kumar**
**Assistant Professor**
**Department of IT**

**Head of the Department**
**Dr. ArunaRao S L**
**Professor and HOD**
**Department of IT**

**External Examiner**

## Acknowledgements

We would like to express our sincere thanks to **Dr. K. V. N. Sunitha, Principal**, BVRIT HYDERABAD, for providing the working facilities in the college.

Our sincere thanks and gratitude to **Dr**. **ArunaRao S L, Head**, Dept. of IT, BVRIT HYDERABAD for all the timely support and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, **Mr. L. Naveen Kumar, Assistant Professor, Department of IT,** BVRIT HYDERABAD for his constant guidance, encouragement and moral support throughout the project.

Finally, We would also like to thank our project coordinator, all the faculty and staff of IT Department who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

**E.Meghana**
**(16WH1A1216)**
**K.Samhitha**
**(16WH1A1229)**
**M.Haritha**
**(16WH1A1247)**

# ABSTRACT

Products and services are heavily discussed on social media, which are conventionally used by brand owners, as well as consumers, to acquire consumer opinions. State-of-the-art opinion mining systems provide summaries of positive and negative sentiment towards a service/product and its various aspects. On a closer look, it is observed that these opinions also contain suggestions, tips and advice, which are often explicitly sought by both brand owners and consumers. Suggestion Mining refers to the task of identifying suggestion expressing sentence in the given text. We study the automatic detection of suggestion expressing words among the opinionated text. The Attention mechanism is used to extract such words that are important to the meaning of the sentence, they enable focusing on specific parts of the input.Various linear methods like CountVectoriser, Bag of words, TFIDF are used in initial classification methods. This project uses Neural Networks like Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and Long short-term memory (LSTM). Using Attention networks, reviews are taken as input and the words which have suggestive sense are highlighted in the output.

# LIST OF FIGURES

# LIST OF TABLES



# LIST OF ABBREVATIONS

| Word | Abbrebvation |
|---|---|
| LSTM | Long Short Term Memory |
| RNN | Recurrent Neural Network |
| R2NN | Recursive Recurrent Neural Network |
| CNN | Convolutional Neural Network |
| HAN | Hierarchial Attention Network |
| GRU | Gated Recurrent Networks |
| NMT | Neural Machine Translation |

# TABLE OF CONTENTS

# 1. INTRODUCTION

## 1.1 OBJECTIVE

Text classification is one of the fundamental tasks in Natural Language Processing. Traditional approaches of text classification represent documents with sparse lexical features such as n-grams and then use a linear model or kernel methods on this representation. More recent approaches used deep learning, such as convolutional neural networks and recurrent neural networks, based on long short-term memory (LSTM) to learn text representations. A large number of documents are being generated all over the world everyday and as a result, automatic text classification has become an essential tool for searching, retrieving, and managing the text representation.

Suggestion mining can be defined as the process of identifying and extracting sentences from unstructured text that contains suggestions. Suggestions in the form of unstructured text could be found in various social media platforms, discussion forums, review websites and blogs. They are often expressed in the form of advice, tips, recommendations, warnings, things to do and various other forms in an explicit as well as an implicit way.

Identifying and retrieving suggestions from the text can be useful in an industrial setting for enhancing a product, summarizing opinions of the consumers, giving recommendations and as an aid in the decision making process. For regular users of online platforms, it could help in seeking advice related to general topics of interest like travel, health, food, shopping, education and many more. Given the abundance of textual information on the Internet about a variety of topics, suggestion mining is undoubtedly a useful task interesting to researchers working in academia as well as industry.

## 1.2 PROBLEM IN EXISTING SYSTEM

Suggestion mining is increasingly becoming an important task, along with sentiment analysis. In today's cyberspace world, people not only express their sentiments and dispositions towards some entities or services, but they also spend considerable time sharing their experiences and advice to fellow customers and the product/service providers with two-fold agenda helping fellow customers who are likely to share a

similarexperienceand motivating the producer to bring specific changes in their offerings which would be more appreciated by the customers. In the existing system, whether the review has any suggestion or not is expressed.

### 1.3 SOLUTION

We evaluate the performance of our proposed model on a benchmark customer review dataset, comprising of the reviews. Our proposed approach shows highlighting the words in the review that have suggestive sense.

### 1.4 FEATURES

Suggestion forums are dedicated forums that are used to provide suggestions for improvement in an entity. The data is collected from feedback posts on the Universal Windows Platform. Often people tend to provide the context in suggestion posts, which gets repetitive in the case of a large number of posts under the same topic. Suggestion mining can act as an automatic summarization in this use case by identifying the sentences where a concrete suggestion is expressed. We observe that the datasets derived from this domain contain a relatively more significant number of positive class instances, as compared to the other domains.

# 2. LITERATURE SURVEY

## 2.1 MACHINE TRANSLATION USING DEEP LEARNING

Now-a-days DNN is playing a significant role in the machine learning technique. Recursive recurrent neural network (R2NN) is the best technique for machine learning. It is the combination of recurrent neural networks and recursive neural networks (such as Recursive autoencoder). This project presents how to train the recurrent neural network for reordering for the source to the target language by using Semi-supervised learning methods. Word2vec tool is required to generate word vectors of the source languageand autoencoder helps us in the reconstruction of the vectors for target language in the tree structure. Results of word2vec play an essential role in the word alignment of the input vectors. RNN structure is very complicated and to train the large data file on word2vec is also a time-consuming task. Hence, powerful hardware support (GPU) is required. GPU improves system performance by decreasing the training time.

The term Machine Translation is used in the sense of translation of one language toanother, with no human improvement. Unlike the traditional statistical machine translation, the neural machine translation aims at building a single neuralnetwork that can be jointly tuned to maximize the translation performance. The models proposed forneural machine translation often belong to a family of encoder decoders and encode a source sentence into a fixed length vector from which a decoder generates a translation.

## 2.2 CONVOLUTIONAL NEURAL NETWORKS

This project reports on a series of experiments with convolutional neural networks (CNN) trained on top of pre-trained word vectors for sentence-level classification tasks. They show that a simple CNN with little hyperparameter tuning and static vectors achieves excellent results on multiple benchmarks. Learning task-specific vectors through fine-tuning offers further gains in performance. They additionally propose a simple modification to the architecture to allow for the use of both tasks specific and static vectors.

In this project,they have described a series of experiments with convolutional neural networks built on top of word2vec. Despite little tuning of hyperparameters, a simple CNN with one layer of convolution performs remarkably well.

## 2.3 HIERARCHICAL ATTENTION NETWORKS

The GRU uses a gating mechanism to track the state of sequences without using separate memory cells. There are two types of gates: the reset gate $r_t$ and the update gate $z_t$. They together control how information is updated to the state.

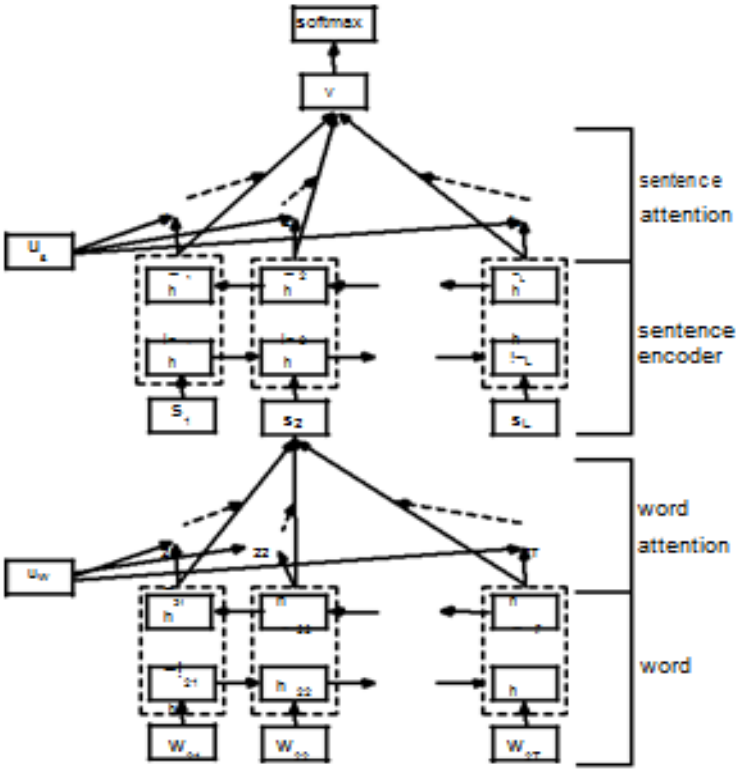


**Figure 1: Hierarchical Attention Network.**

At time t, the GRU computes the new state, this is a linear interpolation between the previous state ht−1 and the current new state ht computed with new sequence information. The gate zt decides how much past information is kept and how much new information is added. Where xt is the sequence vector at time t. The candidate state ht is computed in a way similar to a traditional Recurrent neural network (RNN). Here rt is the reset gate

which controls how much the past state contributes to the candidate state. If rt is zero, then it forgets the previous state.

This project proposesa hierarchical attention network for document classification. Themodel hastwo distinctive characteristics: (i) it has a hierarchical structure that mirrors the hierarchical structure of documents; (ii) it has two levels of attention mechanisms applied at the word and sentence-level, enabling it to attend differentially to more and less important content when constructing the document representation. Experiments conducted on six large scale text classification tasks demonstrate that the proposed architecture outperforms previous methods by a substantial margin. Visualization of the attention layers illustrates that the model selects qualitatively informative words and sentences.

# 3. REQUIREMENT SPECIFICATIONS

Software requirements deal with software and hardwarerequirements deal with resources that need to be installed on a server that provides optimal functioning for the application. These software and hardware requirements need to be installed before the packages are installed. These are the most common set of requirements defined by any operating system. These software and hardware requirements provide compatible support to the operating system in developing an application.

### 3.1.1 SOFTWARE REQUIREMENTS

The software requirements specify the use of all required software products like the data management system. The required software product specifies the numbers and versions. Each interface specifies the purpose of the interfacing software as related to this software product.

- Operating System : Windows 7/10
- Coding Language : Python 3
- Framework : Keras
- Backend : Tensorflow
- IDE : Jupyter Notebook/ Google Colab

### 3.1.2. HARDWARE REQUIREMENTS

The hardware requirement specifies each interface of the software elements and the hardware elements of the system. These hardware requirements include configuration characteristics.

- System : Pentium IV 2.4 GHz
- Hard Disk : 100 GB
- Monitor : 15 VGA Color
- RAM : 2 GB

# 4. DESIGN OF THE SYSTEM

There are several architectures of LSTM units. A common architecture is composed of a cell (the memory part of the LSTM unit) and three regulators, usually called gates, of the flow of information inside the LSTM unit: an input gate, an output gate and a forget gate.

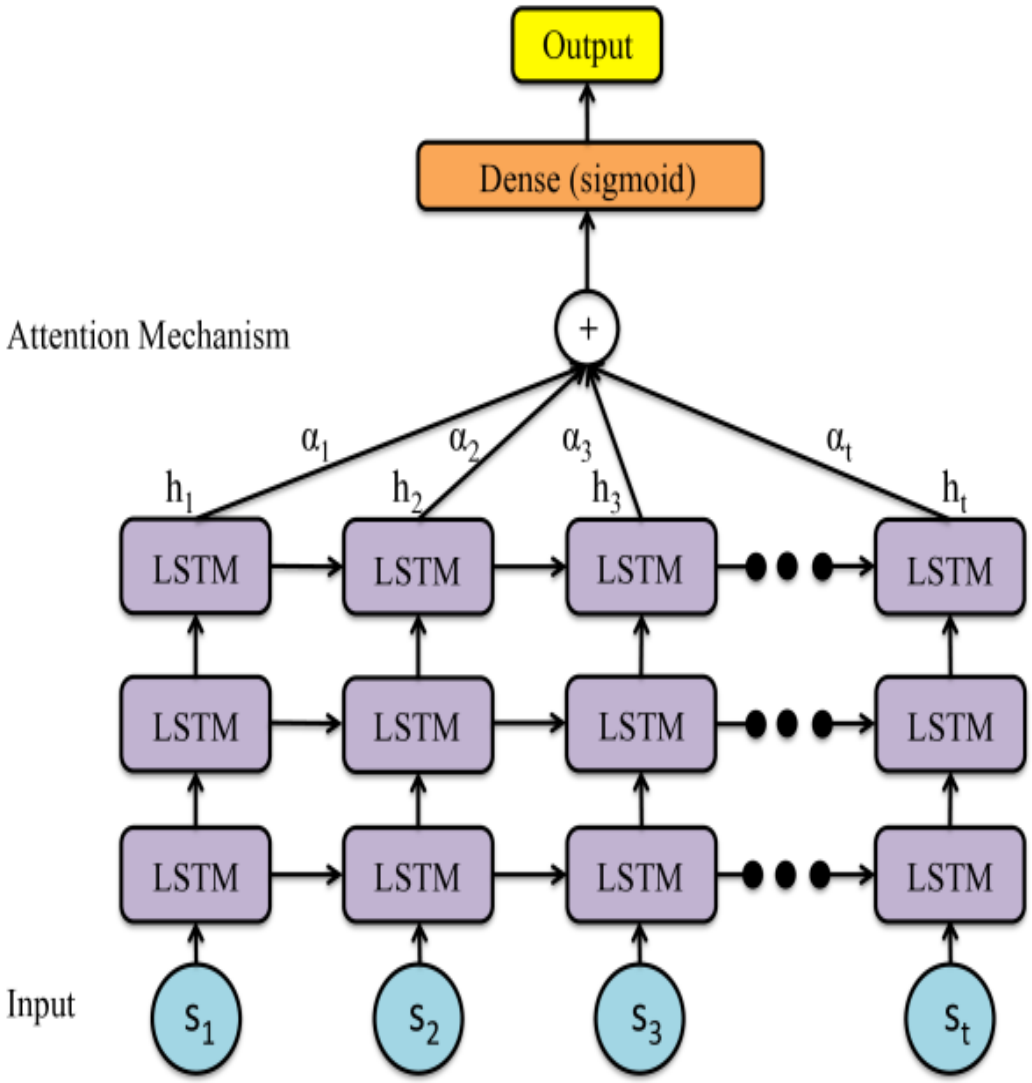


**Figure 2: Architecture diagram of a model with LSTM and Attention layers**

Some variations of the LSTM unit do not have one or more of these gates or mayhave other gates. For example, gated recurrent units (GRUs) do not have an output gate.

Intuitively, the cell is responsible for keeping track of the dependencies between the elements in the input sequence. The input gate controls the extent to which a new value flows into the cell, the forget gate controls the extent to which a value remains in the cell and the output gate controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit. The activation function of the LSTM gates is often the logistic sigmoid function.

There are connections into and out of the LSTM gates, a few of which are recurrent. The weights of these connections, which need to be learned during training, determine how the gates operate.

The RNN using LSTM units can be trained in a supervised fashion, on a set of training sequences, using an optimization algorithm, like gradient descent approach, combined with back-propagation through time to compute the gradients needed during the optimization process, to change each weight of the LSTM network in proportion to the derivative of the error (at the output layer of the LSTM network) with respect to corresponding weight.

A problem with using gradient descent for standard RNN's is that error gradients vanish exponentially quickly with the size of the time lag between important events.

However, with LSTM units, when error values are back-propagated from the output layer, the error remains in the LSTM unit's cell. This "error carousel" continuously feeds error back to each of the LSTM unit's gates, until they learn to cut off the value.

**Text Classification Using Convolutional Neural Network (CNN)**

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

CNN's are generally used in computer vision however,they have recently been applied to various NLP tasks. Let us briefly see what happens when we use CNN on text data through a diagram.The result of each convolution will fire when a special pattern is detected. By varying the size of the kernels and concatenating their outputs, we are allowingitself to detect patterns of multiples sizes (2, 3, or 5 adjacent words).Patterns could be expressions (word n-grams?) like “I hate”, “very good” and therefore, CNN’s can identify them in the sentence regardless of their position.

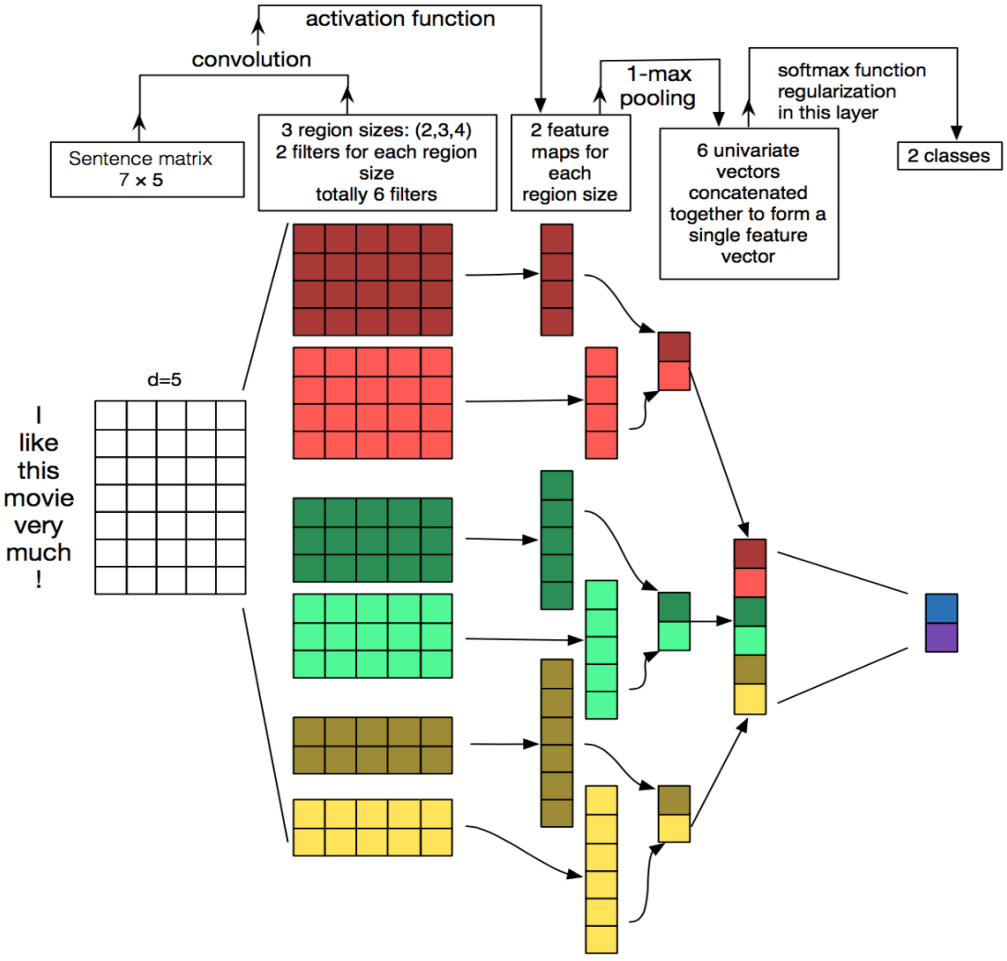


**Figure 3: Architecture diagram of CNN**

The GloVe is an unsupervised learning algorithm for obtaining vector representations for words. 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.

Thisis the flow of a simple CNN model.

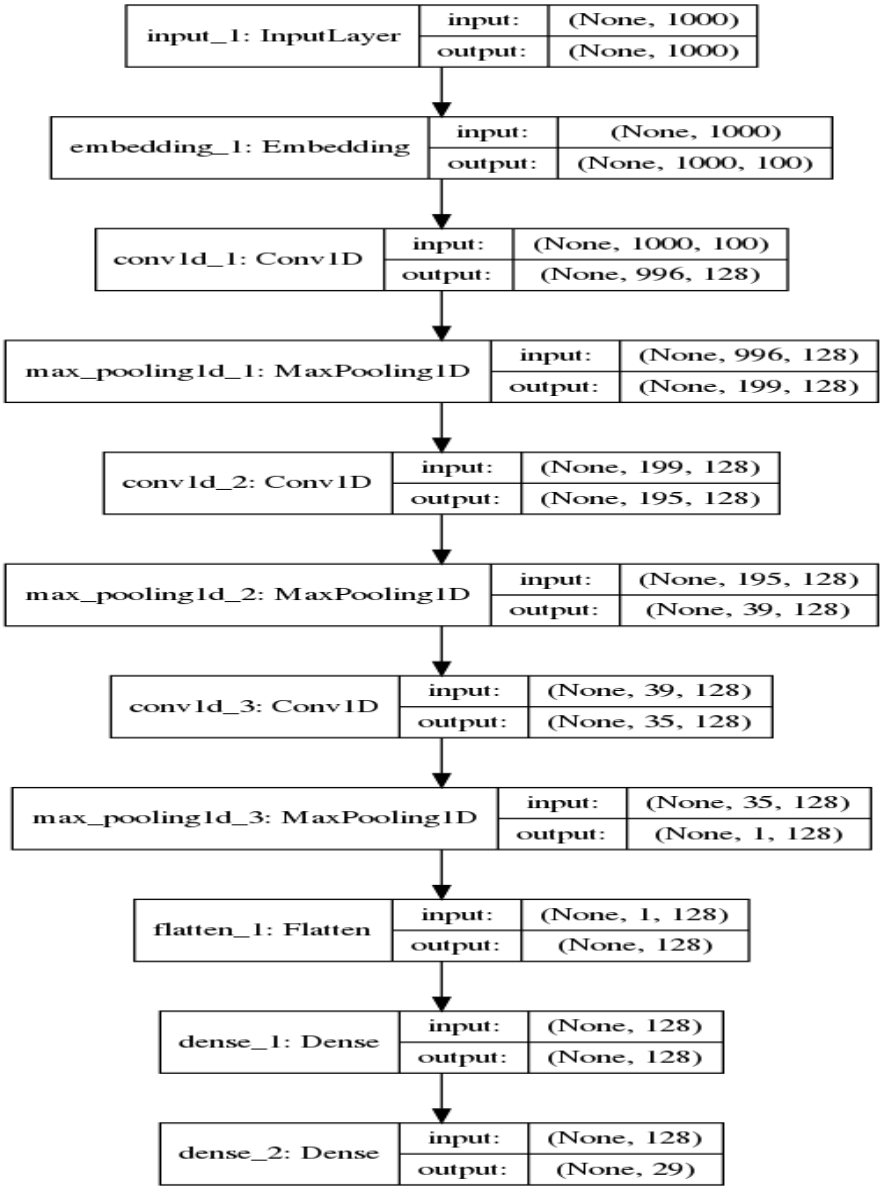


**Figure 4: Flow of a regular CNN model**

**Text Classification Using Recurrent Neural Network (RNN) :**

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a sequence. This allows it to exhibit dynamic temporal behavior for a time sequence.

Using the knowledge from an external embedding, one can enhance the precision of RNN because it integrates new information (lexical and semantic) about the words, information that has been trained and distilled on a very large corpus of data.The pre-trained embedding which we will be using is GloVe.

RNN is a sequence of neural network blocks that are linked to each other like a chain.

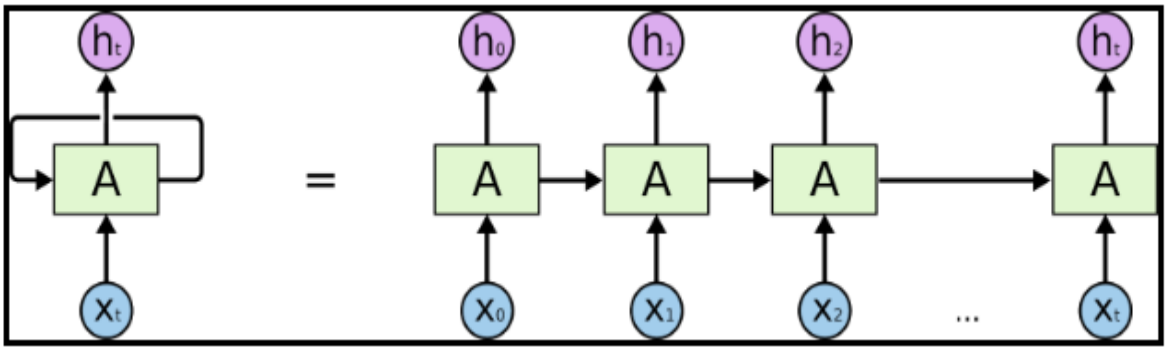


**Figure 5: Example of simple RNN model**

We will process text data, which is a sequence type. The order of words is very important to the meaning. Hopefully, RNN's take care of this and can capture long-term dependencies.

To use Keras on text data, we first have to preprocess it. For this, we can use the KerasTokenizer class. This object takes an argument num_words, which is the maximum number of words kept after tokenization based on their word frequency.

MAX_NB_WORDS=20000

tokenizer = Tokenizer (num_words=MAX_NB_WORDS) tokenizer.fit_on_texts(texts)

Once the tokenizer is fitted on the data, we can use it to convert text strings to sequences of numbers. These numbers represent the position of each word in the dictionary (think of it as mapping).

In this, we will try to tackle the problem by using recurrent neural network and attention-based LSTM encoder.By using the LSTM encoder, we intend to encode all the information of text in the last output of the Recurrent Neural Network before running a feed-forward network for classification.

Other than the regular LSTM, here we have used bidirectional LSTM and concatenate both of the last outputs of LSTM outputs.Keras has provided a very nice wrapper called bidirectional**,** which will make this coding exercise effortless.

These are the steps of the RNN Model.

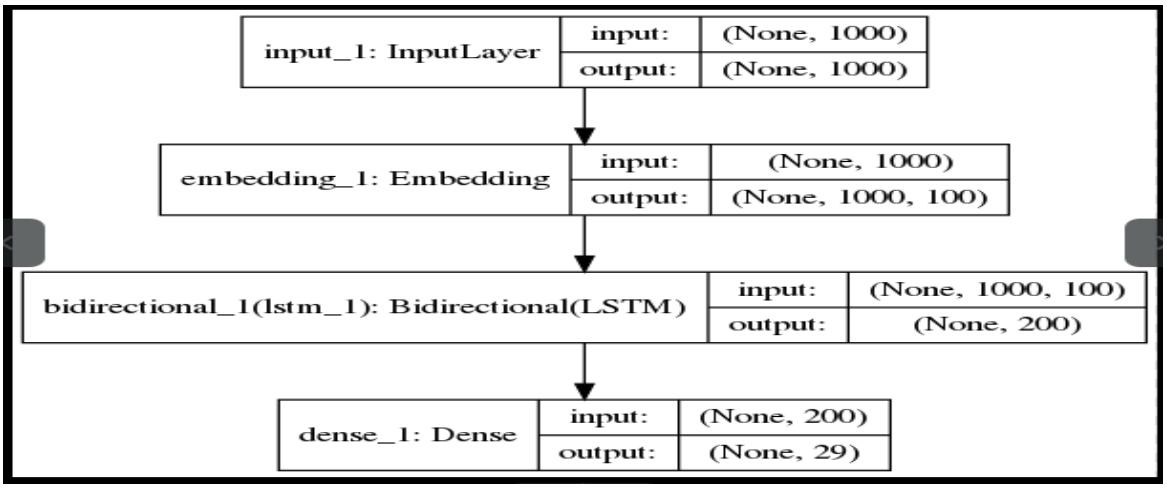


**Figure 6:Basic Flow of a simple model with Bi-Directional LSTM layer**

**Attention and Memory in Deep Learning and NLP**

Attention Mechanisms in Neural Networks are (very) loosely based on the visual attention mechanism found in humans. Human visual attention is well-studied and while there exist different models, all of them primarily come down to being able to focus on a specific region of an image with high resolution while perceiving the surrounding image in low resolution and then adjusting the focal point over time.

Attention in Neural Networks has a long history, particularly in image recognition. But only recentlyattention mechanisms made their way into recurrent neural network

architectures that are typically used in NLP (and increasingly also in vision).

Traditional Machine Translation systems typically rely on sophisticated feature engineering based on the statistical properties of text. In short, these systems are complex, and a lot of engineering effort goes into building them. Neural Machine Translation(NMT) systems work a bit differently. In NMT, we map the meaning of a sentence into a fixed-length vector representation and then generate a translation based on that vector. Instead of relying on things like n-gram counts and trying to capture the higher-level meaning of a text, NMT systems generalize to new sentences better than many other approaches. Perhaps more importantly, NMT systems are much easier to build and train, and they do not require any manual feature engineering.

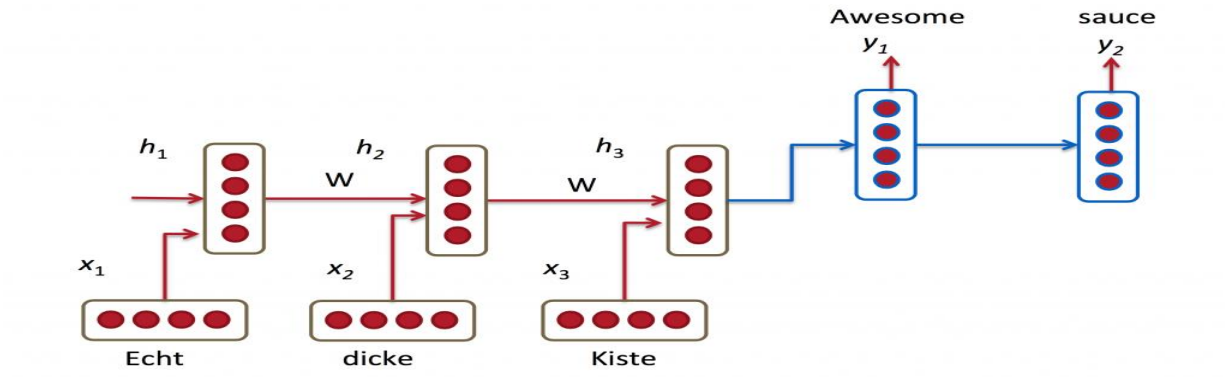


**Figure 7: Example of a Machine Translation System**

In the picture above, “Echt”, “Dicke” and “Kiste” words are fed into an encoder, and after a special signal (not shown) the decoder starts producing a translated sentence. The decoder keeps generating words until a special end of sentence token is produced. Here, the $h$ vectors represent the internal state of the encoder.

If we look closely, we can see that the decoder is supposed to generate a translation solely based on the last hidden state ($h_3$ above) from the encoder. This $h3$vector must encode everything we need to know about the source sentence. It must fully capture its meaning. In more technical terms, that vector is a sentence embedding.

The Attention mechanism in Deep Learning is based on the concept of directing your focus, and it pays greater attention to certain factors while processing the data.In broad terms, attention is one component of a network's architecture, and is in charge of managing and quantifying the interdependence**:**

- Between the input and output elements (General Attention)
- Within the input elements (Self-Attention)

Let ustake an example of how attention works in a translation task. Say we have the sentence "How was your day", which we would like to translate to the French version - "Comment se passe ta journée". For each word in the output sentence, the Attention component of the network will map the important and relevant words from the input sentence and assign higher weights to these words, enhancing the accuracy of the output prediction.

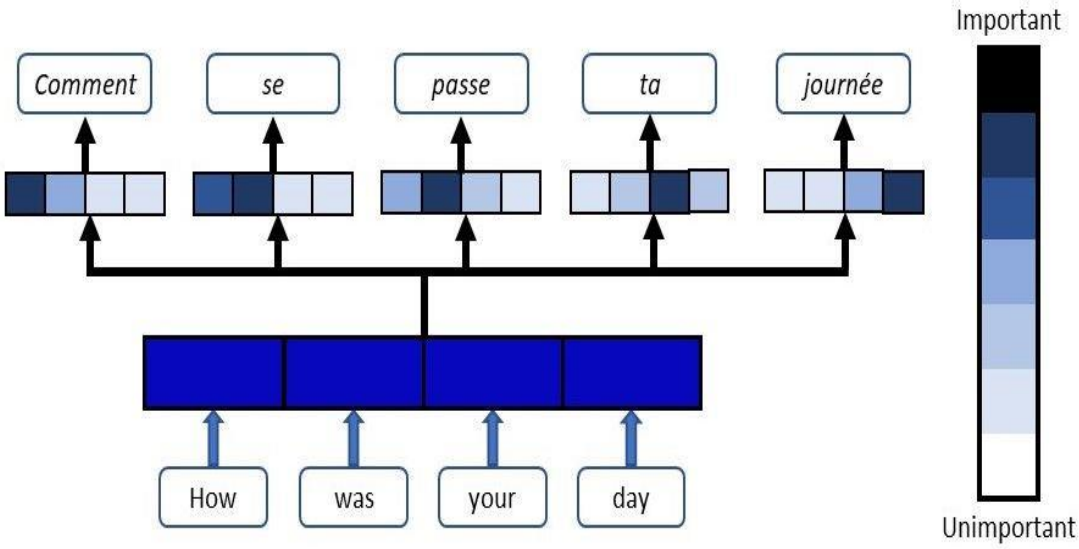


**Figure 8: Assignment of weights to input words**

While attention has its applications in other fields of deep learning such as Computer Vision, its main breakthrough and success come from its applications in Natural Language Processing (NLP) tasks. This is because attention was introduced to address the problem of long sequences in Machine Translation, which is also a problem for most of the NLP tasks as well.

**Attention in Sequence-to-Sequence Models**

Most of the Attention Mechanisms will use the example of sequence-to-sequence (seq2seq) models to explain how it works. This is because attention was originally introduced as a solution to address the main issue surrounding sequence-to-sequence models and to great success.

The standard sequence-to-sequence model is generally unable to accurately process long input sequences since only the last hidden state of the encoder RNN is used as the context vector for the decoder. On the other hand, the Attention Mechanism directly addresses this issue as it retains and utilizes all the hidden states of the input sequence during the decoding process.It does this by creating a unique mapping between each time step of the decoder output to all the encoder hidden states. This means that for each output that the decoder makes, it hasaccess to the entire input sequence and can selectively pick out specific

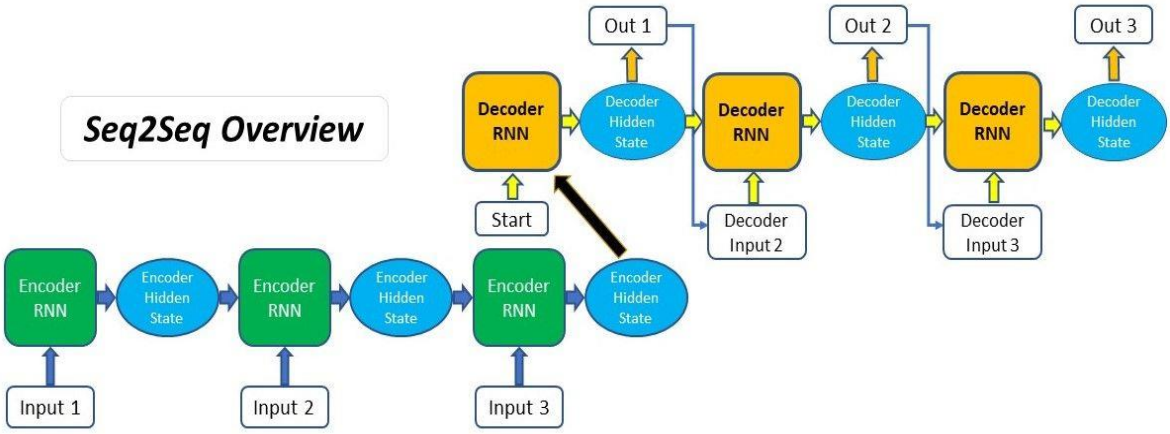


**Figure 9: Sequence-to-Sequence Model**

elements from that sequence to produce the output. Therefore, this mechanism allows the model to focus and place more “Attention” on the relevant parts of the input sequence as needed.

1. **Encoder Hidden States** - Encoder produces hidden states of each element in the

input sequence.

2. **Decoder RNN** - The previous decoder hidden state and decoder output is passed through the Decoder RNN to generate a new hidden state for that time step.

3. **Calculating Alignment Scores** - Using the new decoder hidden state and the encoder hidden states, alignment scores are calculated.

4. **Softmaxing the Alignment Scores** - The alignment scores for each encoder hidden state are combined and represented in a single vector and subsequently softmax function is applied.

5. **Calculating the Context Vector** - The encoder hidden states and their respective alignment scores are multiplied to form the context vector.

6. **Producing the Final Output** - The context vector is concatenated with the decoder hidden state generated in step 2 as passed through a fully connected layer to produce a new output

The process (steps 2-6) repeats itself for each time step of the decoder until a token is produced or output is past the specified maximum length.

# 5.MODULES

This is the flow of a deep learning pipeline for text data which contains all the steps to get a higher level of perspective about the whole process.

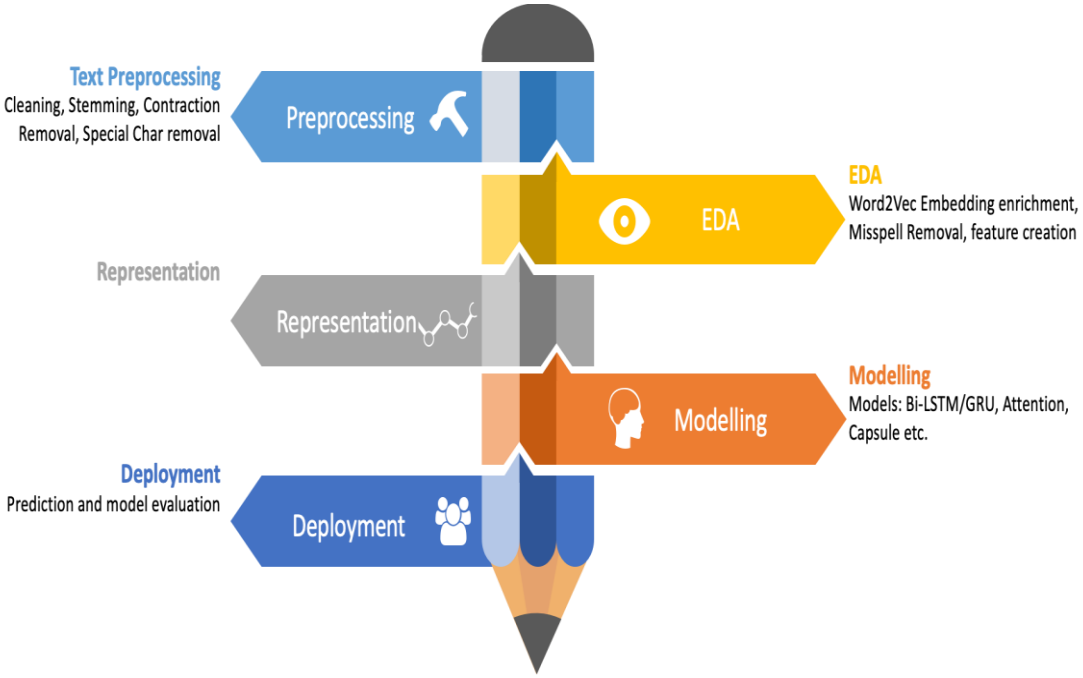


**Figure 10: Deep Learning Pipeline**

We start by cleaning up the text data and performing basic EDA. Here we try to improve our data quality by cleaning up the data. We also try to improve the quality of our gloveembeddings by removing OOV(Out-of-Vocabulary) words.Next, we create a representation for the text that could be fed into a deep learning model. We then start by creating our models and training them. Finally, we evaluate the models using appropriate metrics.

### 5.1.1 Basic Preprocessing Techniques for text data

In most of the cases, we observe that text data is not entirely clean. Data coming from different sources have different characteristics and that makes Text Preprocessing as one of the most important steps in the classification pipeline. For example, text data from Twitter is different from the text data on Quora, or some news/blogging platform, and thus it would need to be treated differently.

**a) Cleaning Special Characters and Removing Punctuations**

Our preprocessing pipeline depends a lot on the gloveembeddingswhich we are going to use for our classification task. In principle, our preprocessing should match the preprocessing that was used before training the word embeddings. Since most of the embeddings do not provide vector values for punctuations and other special characters, the first thing we should do is to get rid of the special characters in text data.

**b) Cleaning Numbers**

**Small Python Trick:** We use an "if" statement in the code below to check whether if a number exists in a text or not. The "if" statement is always fast than a re.sub command and most of our text doesnot contain numbers.

```
defclean_numbers(x):
ifbool(re.search(r'\d', x)):
        x =re.sub('[0-9]{5,}', '#####', x)
        x =re.sub('[0-9]{4}', '####', x)
        x =re.sub('[0-9]{3}', '###', x)
        x =re.sub('[0-9]{2}', '##', x)
return x
```

**Figure 11: Code for Cleaning Numbers**

**d) Removing Contractions**

Contractions are words that we write with an apostrophe. Examples of contractions are words like "ain't" or "aren't". Since we want to standardize our text, it makes sense to expand these contractions.

### 5.1.2 GloVeembeddings

We need to have a way to represent words in a vocab. One way to do that could be to use One hot encoding of word vectors, but that is not a good choice. One of the major reasons is that the one-hot word vectors cannot accurately express the similarity between different words, such as the cosine similarity.

### 5.1.3 Representation: Sequence Creation

One of the things that have made Deep Learning the goto choice for NLP is the fact that we

do not have to hand-engineer features from the text data.

**a) Tokenizer**

In simple words, a tokenizer is a utility function to split a sentence into words. keras.preprocessing.text.Tokenizer tokenizes(splits) the texts into tokens(words) while keeping only the most occurring words in the text corpus.

**b) Pad Sequence**

Normally our model expects that each sequence(each training example) will be of the same length(same number of words/tokens). We can control this using the maxlen parameter.

### 5.1.4 Model

**Using LSTM**

```
Model: "model_7"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_4 (InputLayer)         (None, None)              0
_________________________________________________________________________________________________
embedding_1 (Embedding)      (None, 100, 100)          987100      input_4[0][0]
_________________________________________________________________________________________________
lstm_4 (LSTM)                (None, 100, 5)            2120        embedding_1[3][0]
_________________________________________________________________________________________________
time_distributed_4 (TimeDistrib (None, 100, 1)         6           lstm_4[0][0]
_________________________________________________________________________________________________
softmax_4 (Softmax)          (None, 100, 1)            0           time_distributed_4[0][0]
_________________________________________________________________________________________________
multiply_4 (Multiply)        (None, 100, 100)          0           embedding_1[3][0]
                                                                   softmax_4[0][0]
_________________________________________________________________________________________________
lambda_4 (Lambda)            (None, 100)               0           multiply_4[0][0]
_________________________________________________________________________________________________
dense_8 (Dense)              (None, 1)                 101         lambda_4[0][0]
=================================================================================================
Total params: 989,327
Trainable params: 989,327
Non-trainable params: 0
```



**Figure 12: Summary of a model with LSTM and Time Distributed layers**

LSTMstands for Long Short-Term Memory, which is a particular type of recurrent neural

network. An LSTM has a similar control flow as a recurrent neural network. It processes data passing on information as it propagates forward. To add new information, it transforms the existing information completely by applying a function. Because of this, the entire information is modified, on the whole. They make small modifications to the information by multiplications and additions. With LSTM's, the information flows through a mechanism known as cell states. This way, they can selectively remember or forget things.

**Using Attention, LSTM and Time Distributed Layer:**

```
Model: "model_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         (None, None)              0
_________________________________________________________________
embedding_1 (Embedding)      (None, 100, 100)          965700
_________________________________________________________________
lstm_10 (LSTM)               (None, 100, 128)          117248
_________________________________________________________________
lstm_11 (LSTM)               (None, 100, 64)           49408
_________________________________________________________________
dropout_7 (Dropout)          (None, 100, 64)           0
_________________________________________________________________
lstm_12 (LSTM)               (None, 100, 32)           12416
_________________________________________________________________
time_distributed_4 (TimeDist (None, 100, 1)            33
_________________________________________________________________
dropout_8 (Dropout)          (None, 100, 1)            0
_________________________________________________________________
softmax_4 (Softmax)          (None, 100, 1)            0
_________________________________________________________________
attention_with_context_4 (At (None, 1)                 3
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 2
=================================================================
Total params: 1,144,810
Trainable params: 1,144,810
Non-trainable params: 0
```

**Figure 13: Summary ofmodel with LSTM, Attention and Time Distributed layers**

LSTM with Time Distributed Layer allows the problem to be framed and learned as it was

defined, which is one input to one output, keeping the internal process for each time step separate. It also simplifies the network by requiring far fewer weights such that only a one-time step is processed at a time.Instead of having several input models, the time distributed layer allows the use of only one model and apply it to each input. Attention networks allow the decoder to pay attention to different parts of the source sequence at different decoding steps to attain more number of features.The gates present in LSTM can learn which data in a sequence is important to keep or throw away. By doing that, it can pass relevant information down the long chain of sequences to make predictions.

**Using Bi-Directional LSTM and Attention Layer**

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, None)              0
_________________________________________________________________
embedding_1 (Embedding)      (None, 100, 100)          965700
_________________________________________________________________
bidirectional_5 (Bidirection (None, 100, 256)          234496
_________________________________________________________________
bidirectional_6 (Bidirection (None, 100, 128)          164352
_________________________________________________________________
dropout_3 (Dropout)          (None, 100, 128)          0
_________________________________________________________________
attention_with_context_2 (At (None, 128)               16640
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 65
=================================================================
Total params: 1,389,509
Trainable params: 1,389,509
Non-trainable params: 0
```

**Figure 14: Summary of a model with Bi-Directional LSTM layer**

The basic idea of bidirectional recurrent neural networks is to present each training sequence forwards and backward to two separate recurrent networks, both of which are

connected to the same output layer.They utilize both the previous and future context by processing the data from two directions with two separate hidden layers. One layer processes the input sequence in the forward direction, while the other processes the input in the reverse direction. The output of the current time step is then generated by combining both layers hidden vector. This means that for every point in a given sequence, the Bi-Directional Recurrent Neural Network has complete, sequential information about all points before and after it. Also, because the network is free to use as much or as little of this context as necessary, there is no need to find a time-window or target delay size.

### 5.1.5 Deployment

We evaluate all the above models using large datasets containing reviews. Based on the performance of each model, we can conclude which model works efficiently. The performance evaluation is done using metrics such as Accuracy, Precision, Recall and F1 score. Accuracy is the percentage of texts that were predicted with the correct tag. Precision is the percentage of examples the classifier got right out of the total number of examples that is predicted for a given tag. The Recall is the percentage of examples the classifier predicted for a given tag out of the total number of examples it should have predicted for that given tag. And F1 Score is the harmonic mean of precision and recall.

# 7. IMPLEMENTATION

## PREPROCESSING

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. When creating a machine learning model, it is not always a case that we come across the clean and formatted data. Moreover, while doing any operation with data, it is mandatory to clean it and put it in a formatted way. So for this, we use data preprocessing tasks.

```
import pandas as pd
from nltk.corpus import stopwords
import numpy as np
from keras.preprocessing.sequence import pad_sequences
import tensorflow as tf
import keras.backend as K
from keras.preprocessing.text import Tokenizer, text_to_word_sequence
from keras.layers import GRU, LSTM, Bidirectional, Conv1D
```

```
from numpy import array
import re
def decontracted(phrase):
    phrase = re.sub("won\'t", "will not", phrase)
    phrase = re.sub("can\'t", "cannot", phrase)
    phrase = re.sub("n\'t", " not", phrase)
    phrase = re.sub("\'re", " are", phrase)
    phrase = re.sub("\'s", " is", phrase)
    phrase = re.sub("\'d", " would", phrase)
    phrase = re.sub("\'ll", " will", phrase)
    phrase = re.sub("\'t", " not", phrase)
    phrase = re.sub("\'ve", " have", phrase)
    phrase = re.sub("\'m", " am", phrase)
    return phrase
df = pd.read_csv("/content/drive/My Drive/Colab Notebooks/dataset.csv")
print(df['review'][8498])
df['review'] = list(decontracted(df['review'][phrase]) for phrase in range(len(df['review'])))
df['review'] = df['review'].str.replace('[^a-zA-Z.\ ]', ' ')
docs = df['review'].str.lower()
```

**Figure 15: Code for Preprocessing**

**Before Preprocessing:**

The reviews which contain raw data before preprocessing are shown below

```
45] from numpy import array
    df = pd.read_csv("/content/drive/My Drive/train - train.csv")
    print("Before pre-processing")
    print(df['review'][8464])

    Before pre-processing
    its not possible to set the hijri adjustment for the month of ramadan
```

**Figure 16(a): Review before Preprocessing**

**After Preprocessing:**

The reviews which contain formatted data after preprocessing are shown below

```
[46] docs = df.apply(lambda x: clean(x.review), axis=1)
     print("After pre-processing")
     print(docs[8464])

     After pre-processing
     it is not possible to set the hijri adjustment for the month of ramadan
```

**Figure 16(b): Review after Preprocessing**

**EMBEDDINGS**

An embedding is a relatively low-dimensional space into which we can translate high-dimensional vectors. Embeddings make it easier to do machine learning on large inputs like sparse vectors representing words. Ideally, an embedding captures some of the semantics of the input by placing semantically similar inputs close together in the embedding space. An embedding can be learned and reused across models.

```
embeddings_index = {}
missing_words = []
f = open('/content/drive/My Drive/glove.6B.100d.txt', encoding="utf8")
for line in f:
    try:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
    except:
        print(word)
        pass
f.close()
embed_size = 100
embedding_matrix = np.zeros((len(word_index) + 1, embed_size))
absent_words = 0
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
    else:
        absent_words += 1
        missing_words.append(word)
print(missing_words)
print(len(missing_words))
print(len(word_index))
print('Total absent words are', absent_words, 'which is', "%0.2f" % (absent_words * 100 / len(word_index)), '% of total wo
```



**Figure 17: Code for Embeddings**

## ATTENTION WITH CONTEXT

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

The way a query and a key are combined to get a value is by doing a weighted sum. On the other hand, the “Convolutional sequence to sequence learning” and the “Attention is all

you need" paper, which achieved state-of-the-art performances on machine translation, uses an attention mechanism where the dot product of a query and a key is taken to get a value.

```
class AttentionWithContext(Layer):
    def __init__(self,
                 W_regularizer=None, u_regularizer=None, b_regularizer=None,
                 W_constraint=None, u_constraint=None, b_constraint=None,
                 bias=True, **kwargs):

        self.supports_masking = True
        self.init = initializers.get('glorot_uniform')

        self.W_regularizer = regularizers.get(W_regularizer)
        self.u_regularizer = regularizers.get(u_regularizer)
        self.b_regularizer = regularizers.get(b_regularizer)

        self.W_constraint = constraints.get(W_constraint)
        self.u_constraint = constraints.get(u_constraint)
        self.b_constraint = constraints.get(b_constraint)

        self.bias = bias
        super(AttentionWithContext, self).__init__(**kwargs)

    def build(self, input_shape):
        assert len(input_shape) == 3

        self.W = self.add_weight((input_shape[-1], input_shape[-1],),
                                 initializer=self.init,
                                 name='{}_W'.format(self.name),
                                 regularizer=self.W_regularizer,
                                 constraint=self.W_constraint)
```

**Figure 18: Code for Attention with Context**

## MODEL

### Using LSTM

Here the input dataset is fed to the embedding layer after the preprocessing step. The output from the embedding layer is fed to a time distributed layer. Using this layer, instead of having several input models, we can use only one model and apply it to each input. Then

the attention scores are obtained using attention networks and they are normalized using a softmax function. These normalized attention scores tend to form context vector and this is again given as input to a dense layer. A densely connected layer provides learning features from all the combinations of the features of the previous layers by performing linear operations on each input. The final model will include all the above layers as input and performs computations to get the desired output.

```
from keras.models import Model
from keras.layers import *
from keras.preprocessing.sequence import pad_sequences
from keras import regularizers

def create_models():
    inp = Input((None,))
    embs = e(inp)
    lstm = LSTM(5, return_sequences=True)(embs)
    attention = TimeDistributed(Dense(1))(lstm)
    attention_vals = Softmax(axis=1)(attention)
    scaled_vecs = Multiply()([embs,attention_vals])
    context_vector = Lambda(lambda x: K.sum(x,axis=1))(scaled_vecs)
    out = Dense(1,activation='sigmoid')(context_vector)
    model = Model(inp, out)
    model_with_attention_output = Model(inp, [out, attention_vals])
    model.compile(optimizer='adam',loss='binary_crossentropy', metrics=['accuracy'])
    return model, model_with_attention_output

model,model_with_attention_output = create_models()
print(model.summary())

model.fit(pdocs, np.array(labels), validation_split=0.1, epochs = 5)


loss, accuracy = model.evaluate(pdocs,labels , verbose=0)
print('Training Accuracy is {}'.format(accuracy*100))
```

**Figure 19: Code for model with LSTM layer**

**Using LSTM, Attention and Time Distributed Layer**

Similar to the above model, the input dataset is fed to the embedding layer after the preprocessing step. The output from the embedding layer is fed to the LSTM layer.An LSTM allows the preservation of gradients. The memory cell remembers the first input as long as the forget gate is open and the input gate is closed. Then the output is fed

to the dropout layer which prevents overfitting.

```
from keras.models import Model
from keras.layers import *
from keras.preprocessing.sequence import pad_sequences
from keras import regularizers
from keras.utils.vis_utils import plot_model

def create_models():
    inp = Input((None,))
    embs = e(inp)
    lstm_1 = LSTM(128, return_sequences=True)(embs)
    lstm_2 = LSTM(64, return_sequences=True)(lstm_1)
    x = Dropout(0.4)(lstm_2)
    lstm_3 = LSTM(32, return_sequences=True)(x)
    attention = TimeDistributed(Dense(1))(lstm_3)
    drop = Dropout(0.4)(attention)
    attention_vals = Softmax(axis=1)(drop)
    scaled_vecs = Multiply()([embs,attention_vals])
    context_vector = Lambda(lambda x: K.sum(x,axis=1))(scaled_vecs)
    out_att = AttentionWithContext()(attention_vals)

    out = Dense(1,activation='sigmoid')(out_att)
    model = Model(inp, out)
    model_with_attention_output = Model(inp, [out, attention_vals])
    model.compile(optimizer='adam',loss='binary_crossentropy', metrics=['accuracy'])
    return model, model_with_attention_output

model,model_with_attention_output = create_models()
print(model.summary())
```

**Figure 20: Code for model with LSTM and Attention using Time Distributed Layer**

Again the LSTM is introduced with a time distributed layer which helps us to use only one model for each input. A dropout layer is again introduced and then the attention scores are obtained using attention networks and they are normalized using a softmax function. These normalized attention scores tend to form context vector and this is again given as input to a dense layer. A densely connected layer provides learning features from all the combinations of the features of the previous layers by performing linear operations on each input. The final model will include all the above layers as input and performs computations to get the desired output.

**Using Bi-Directional LSTM with Attention Layer**

```
from keras.models import Model
from keras.layers import *
from keras.preprocessing.sequence import pad_sequences
from keras.regularizers import l2 as l2_reg

def create_models():
    maxlen = 100
    inp = Input((None,))
    x = e(inp)
    x = Bidirectional(LSTM(128, return_sequences=True))(x)
    x = Bidirectional(LSTM(64, return_sequences=True))(x)
    x = Dropout(0.4)(x)
    x = AttentionWithContext()(x)
    x = Dense(64, activation="relu")(x)
    x = Dense(1, activation="sigmoid")(x)
    model = Model(inputs=inp, outputs=x)
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model

model = create_models()
print(model.summary())

model.fit(pdocs, np.array(labels),validation_split=0.1, epochs = 25,batch_size=100)


loss, accuracy = model.evaluate(pdocs,labels , verbose=1)
print('Training Accuracy is {}'.format(accuracy*100))
```

**Figure 21: Code for model with Bi-Directional LSTM layer**

In this model, the input dataset is fed to the embedding layer after the preprocessing step. The output from the embedding layer is fed to the Bi-Directional LSTM layer. This layer makes use of the previous context by processing the data in both directions with two separate hidden layers, which are then fed to the same output layer. Then the output is fed to the dropout layer which prevents overfitting. Then the attention scores are obtained using attention networks and they are normalized using a sigmoid function. These normalized attention scores tend to form context vector and this is again given as input to a dense layer. A dense layer provides learning features from all the combinations of the features of the previous layers by performing linear operations on each input. The final model will include all the above layers as input and performs computations to get the desired output.

# 7.TESTING

As we work with datasets, a machine learning algorithm works in two stages. One is the training stage and the other is the testing stage. Under supervised learning, we usually use two different files, one for training the data and the other for testing the data.

| SNo. | Model | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|---|
| 1 | LSTM | 75 | 0.568095 | 0.753721 | 0.647874 |
| 2 | LSTM+Attention+Time Distributed layers | 75 | 0.814374 | 0.753721 | 0.647874 |
| 3 | Bi-LSTM+Attention | 87 | 0.845009 | 0.808235 | 0.722521 |

**Table 1 :Model Comparison**

## Using LSTM layer

We get the following as output when we use LSTM with Attention Networks.

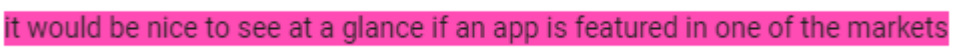


**Figure 22: Output of a model with LSTM layer**

## Using Attention + LSTM + Time Distributed Layers

We get the following as output when we use LSTM with Time Distributed Layer and Attention Networks.

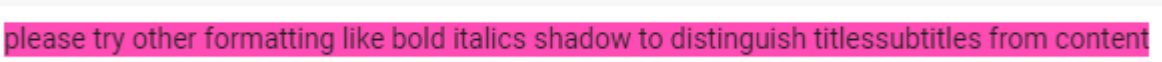


**Figure 23: Output of a model with Attention and LSTM with Time Distributed Layers**

## Using Bi-Directional LSTM with Attention layer

We get the following as output when we use Bi-Directional LSTM with Attention Networks.

### When we use 7 epochs

please enable removing language code from the dev center language history for example if you ever selected ru and ruru laguages and you published this xap to the store then it causes tile localization to show the enusdefault tile localization which is bad

**Figure 24(a): Output of a model with Bi-Directional LSTM layer using 7 epochs**

### When we use 25 Epochs

```
Epoch 16/25
7650/7650 [==============================] - 84s 11ms/step - loss: 0.0136 - acc: 0.9957 - val_loss: 1.0046 - val_acc: 0.8424
Epoch 17/25
7650/7650 [==============================] - 84s 11ms/step - loss: 0.0123 - acc: 0.9957 - val_loss: 0.9236 - val_acc: 0.8388
Epoch 18/25
7650/7650 [==============================] - 84s 11ms/step - loss: 0.0150 - acc: 0.9942 - val_loss: 0.9050 - val_acc: 0.8235
Epoch 19/25
7650/7650 [==============================] - 85s 11ms/step - loss: 0.0118 - acc: 0.9953 - val_loss: 1.0366 - val_acc: 0.8271
Epoch 20/25
7650/7650 [==============================] - 85s 11ms/step - loss: 0.0080 - acc: 0.9969 - val_loss: 1.2624 - val_acc: 0.8294
Epoch 21/25
7650/7650 [==============================] - 84s 11ms/step - loss: 0.0086 - acc: 0.9961 - val_loss: 1.2448 - val_acc: 0.8235
Epoch 22/25
7650/7650 [==============================] - 85s 11ms/step - loss: 0.0080 - acc: 0.9962 - val_loss: 0.9853 - val_acc: 0.8388
Epoch 23/25
7650/7650 [==============================] - 84s 11ms/step - loss: 0.0083 - acc: 0.9961 - val_loss: 1.3112 - val_acc: 0.8318
Epoch 24/25
7650/7650 [==============================] - 84s 11ms/step - loss: 0.0095 - acc: 0.9967 - val_loss: 1.4289 - val_acc: 0.8400
Epoch 25/25
7650/7650 [==============================] - 84s 11ms/step - loss: 0.0090 - acc: 0.9956 - val_loss: 1.2718 - val_acc: 0.8435
8500/8500 [==============================] - 27s 3ms/step
Training Accuracy is 98.16470587954802
```

please enable removing language code from the dev center language history for example if you ever selected ru and ruru laguages and you published this xap to the store causes tile localization to show the enusdefault tile localization which is bad

**Figure 24(b): Output of a model with Bi-Directional LSTM layer using 25 epochs**

Therefore, from the above-provided outputs, we can understand that the model using Bi-Directional LSTM with Attention networks works efficiently than the other models.

# 8.CONCLUSION AND FUTURE SCOPE

## 8.1 CONCLUSION

In addition to the positive and negative sentiments expressed by speakers, opinions on the web also convey suggestions. Such text comprises advice, recommendations and tips on a variety of points of interest. We propose that suggestions can be extracted from the available opinionated text and put to several use cases. As a convenient side-effect we obtained better visualization using the highly informative components of a document. Our model progressively builds a document vector by aggregating important words into sentence vectors and then aggregating important sentence vectors to document vectors. Experimental results demonstrate that our model performs significantly better than previous methods. Visualization of these attention layers illustrates that our model is effective in picking out important words and sentences.

## 8.2 FUTURE SCOPE

The future scope of the project is that, to try improving the model performance by tuning in some hyperparameters and working with a larger dataset.

## 9. REFERENCES

1. Yang, Zichao, et al. “Hierarchical attention networks for document classification” Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies. 2016.

2. Anand, Sarthak, et al “Suggestion Mining from Online Reviews using ULMFiT” arXiv preprint arXiv:1904.09076(2019).

3. Liu, Gang, and JiabaoGuo. “Bidirectional LSTM with attention mechanism and convolutional layer for text classification” Neurocomputing 337 (2019): 325-338.

4. Vaswani, Ashish, et al. “Attention is all you need” Advances in neural information processing systems. 2017.

5. Brun, Caroline, and Caroline Hagege. “Suggestion Mining: Detecting Suggestions for Improvement in Users Comments” Research in Computing Science 70.79.7179 (2013): 5379-62.

6. Negi, Sapna, and P. Buitelaar. “Suggestion mining from opinionated text” Sentiment Analysis in Social Networks(2017): 129-139.

7. Viewing text through the eyes of a machine – Towards Data Science - https://towardsdatascience.com/viewing-text-through-the-eyes-of-a-machine-db30c744ee17

8. Using Keras, Preprocessing data, Model Creation – Machine learning Mastery - machinelearningmastery.com

9. Natural Language Processing (NLP) - https://mlwhiz.com/nlpseries/