Abstractive Text Summarizer Using Deep Learning

Mukund Kulkarni, Abu Dujana.

Department of Computer Science and Engineering, Vishwakarma Institute of Technology, Pune.

Abstract *— In this era, there is tremendous amount of information available on the internet. Some of them stored in the form of numbers, texts, images, etc. This particular project emphasizes on the abstractive summarization of the huge amount of texts available in the modern platforms. They maybe in form of articles, reviews or descriptions. This project uses the principles of Natural Language Processing (NLP) and Deep Learning to prepare a model that summarizes a given input text into a single short sentence, without extracting any existing sentence from the given input.*

I. INTRODUCTION

Text summarization is a process of extracting important information from original text and presenting that information in form of a summary. In recent times, the need of summarization can be seen in various purposes and in various domains as well, such as news articles summary, business analysis, customer reviews, etc. This helps to retrieve and analyze information with less amount of processing and still get meaningful results.

On the internet, there are many examples available like, news article summarizer such as Microsoft News2, Google1 or Columbia Newsblaster3. BaseLine, FreqDist, SumBasic, MEAD, AutoSummarize & SWESUM are few popular biomedical summarization tools.

In 1950, the first system was invented. The automatic summarizer in general selected important sentences from the document and grouped them together, it consumed less time or saved a lot of time to understand the content within the large document. Extractive summarizers extract important sentences or phrases from the original input and group them to produce a summary without changing the contents of the original text. In Abstractive summarizer, the model is first trained using a lot of training articles, then takes an input and finally generates a summary on it's own, the summary is entirely generated without copying any sentence from the input.

II. LITERATURE REVIEW

Saranyamol C S NP et.al [1] talked about various achievements in the field of Natural Language Processing (NLP). He also wrote about methods that do deeper analysis of text. Deepali K. Gaikwad NP et.al [10] Explained the basic working of different algorithms, their advantages and disadvantages. The author also explained the key tools and techniques required to assemble any particular algorithm. Dixit Rucha NP et.al [11] emphasized on the mathematics used in various algorithms, their reliability, accuracy, etc. The authors also introduced their own algorithm which consisted of four components: fuzzifier, inference engine, defuzzifier and the fuzzy knowledge base. Mohamed Ahmed NP et.al [12] his paper gave a brief description of document graph summarization and query based summarization. Both methods had complex deep learning algorithms included. Naresh Kumar Nagwani NP et.al [13] discussed about similarity based extraction using different algorithms. The paper also proposed a simple algorithm to summarize text.

***Tools*: -**

a) Anaconda Prompt
b) Python Kernal.
c) Tensorflow.
d) Python libraries.

III. METHODOLOGY

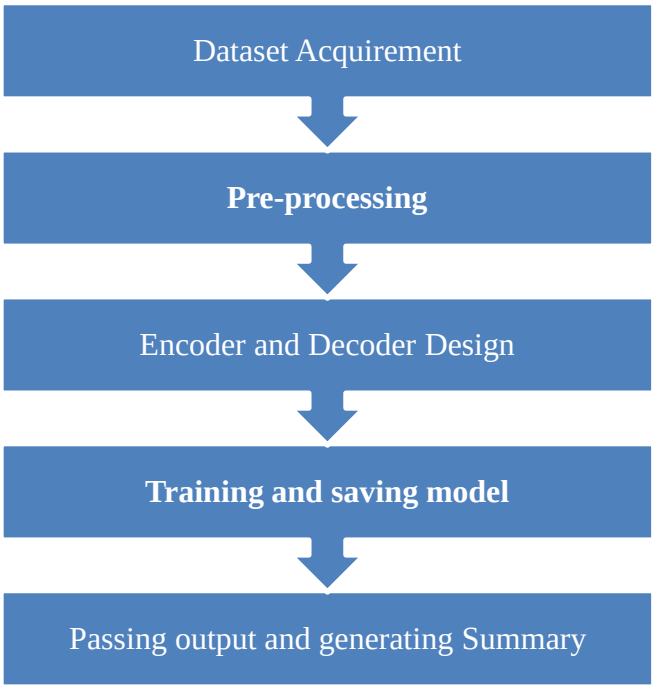

Fig.1 Flowchart

There are various approaches to make an abstractive text summarizer, this is the basic overview of the tasks.

3.1. Dataset Acquirement

The First step is to collect a dataset of articles to train the model. The important factor to be noted is that the entire dataset must contain articles of one single topic, such as amazon customer reviews, restaurant ratings, etc. The more concrete the topic is, the better the model will perform at the end. This unfortunate limitation of this model occurs because if the topic of the model is not pretty concrete, the model then gives out random words as summary, which leads to meaningless summaries.

3.2. preprocessing

After Deciding the dataset, the texts need to undergo a lot of preprocessing, namely :-

1. Cleaning
2. Tokenization
3. Word embedding
4. Vectorization

3.2.1. Cleaning

This step consists of removing punctuation, symbols, links, and short forms from the articles and converting every word into lowercase. It is crucial to train the model with clean texts because the model treats every single word differently. For example, the word "they're" will be treated as different from "they are", hence, this step can't be missed.

3.2.2. Tokenization

Tokenization basically means to separate every word and store it into a list, with quotation marks. For example, the tokenized form of the sentence "go ahead and give" will be:

["go", "ahead", "and", "give"]

3.2.3. Word embedding

This is actually a deep concept of how to actually translate a word to computer language, considering the order, gender, type, etc. Since computers don't understand words, there are various algorithms on how to mathematically translate words to numbers or vectors without losing the meaning of the words for the computer.

3.2.4. Vectorization

Vectorization is the methodology to map words or phrases into corresponding vectors of real numbers.

3.3. Encoders and Decoders:

After the cleaning of the texts, we move to building the encoders and decoders. The prior step to this is building the sequence-to-sequence model. Since the input here is a long sequence of words and the output is a short summary (which is a sequence as well), this is modelled as a Many-to-Many Seq2Seq problem. Below is a sample Seq2Seq model architecture:

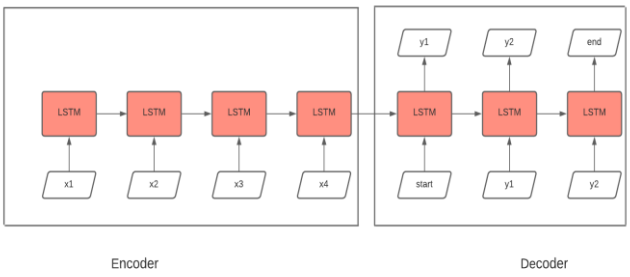

Fig.2 Seq2Seq model

After the Seq2Seq modelling, the encoder and decoder designing comes into picture. Encoder-Decoder architecture is mainly used to solve the

Seq2Seq problems where input and output sequences are of different lengths.

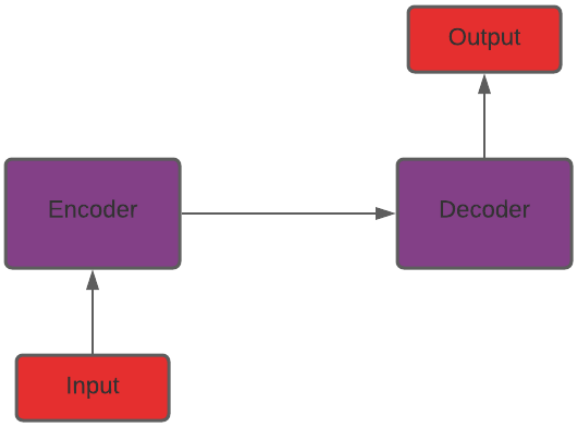

Fig.3 Overall working

Generally, Recurrent Neural Networks (RNNs), Gated Recurrent Neural Networks (GRU) or Long Short Term Memory (LSTM) are preferred as the components of the architecture. This is because they are capable of capturing long term dependencies by tackling the problem of vanishing gradient. The Encoder-Decoder is set up in two phases: Training phase and Inference phase.

3.3.1 Training Phase

In this phase, the model is trained to predict the target sequence offset by one timestep.

a) Encoder: An Encoder LSTM model reads the entire input sequence where at each timestep, one word is fed into the encoder. It processes the information at every timestep and captures the contextual information present in the input sequence.

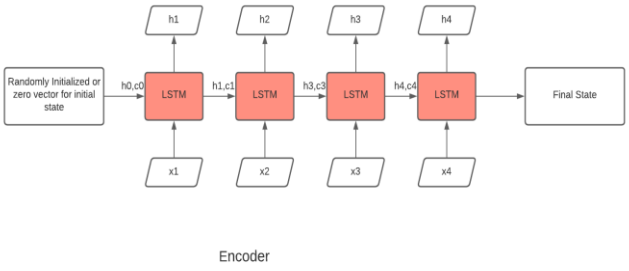

Fig.4 LSTM encoder

b) Decoder: This is also an LSTM network which reads the entire target sequence word to word and predicts the same sequence offset by one timestep. It is trained to predict the next word in the sequence, given the previous word.

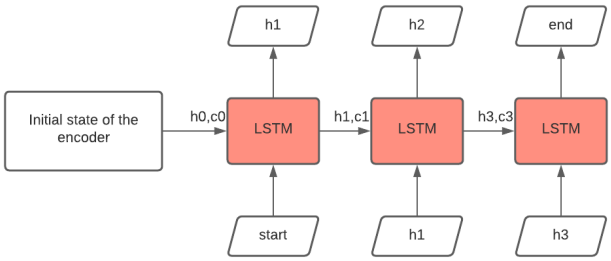

Fig.5 LSTM decoder

<start> and <end> are the special tokens which are added to the target sequence before feeding it into the decoder.

3.3.2 Inference Phase

After training, this model is now tested on new sequences for which the target sequence is unknown.

The steps are:

1. Encode the entire input sequence then initialize the decoder with internal states of encoder
2. Pass the <start> token as input to the decoder.
3. Run the decoder for one timestep.
4. The output of the decoder will be the probability of the next word, the word with highest probability will be selected.
5. Pass the output again as input to decoder.
6. Keep repeating step 3 to 5 until the end token is generated.

In this particular approach, two layered LSTMS have been used, which looks like the figure given below.

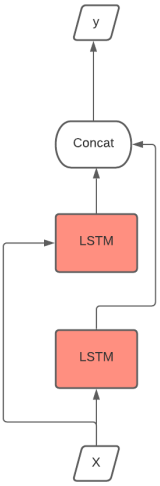

Fig.6 Layered LSTM

IV. LIMITATIONS

4.1 Sequence length

Since the encoder converts the entire input sequence into a fixed length vector and then the decoder predicts the output sequence, this only works for short sequences since the decoder is looking at the entire input sequence for the prediction. It is actually difficult for the encoder to memorize long sequences into a fixed length vector. This limitation is managed by a thing called Attention Mechanism.

4.1.1 Attention Mechanism

Instead of looking all the words in the input sequence, we can increase the importance of specific parts of the input sequences that result in target sequence. For example, consider

Input: "Which sport do you like the most?"

Output: "I love soccer"

The first word 'I' in the output is related to the fourth word 'you' in the input, similarly the second word 'love' is related to the fifth word 'like'. Now there are two different classes of attention mechanism depending on the way the attended context vector is derived.

a) Global Attention
b) Local Attention

In global attention, the attention is placed on all source positions i.e. all the hidden states of the encoder are considered for the attended context vector. In the local attention, only a few hidden states of the encoder are considered for deriving the attended context vector.

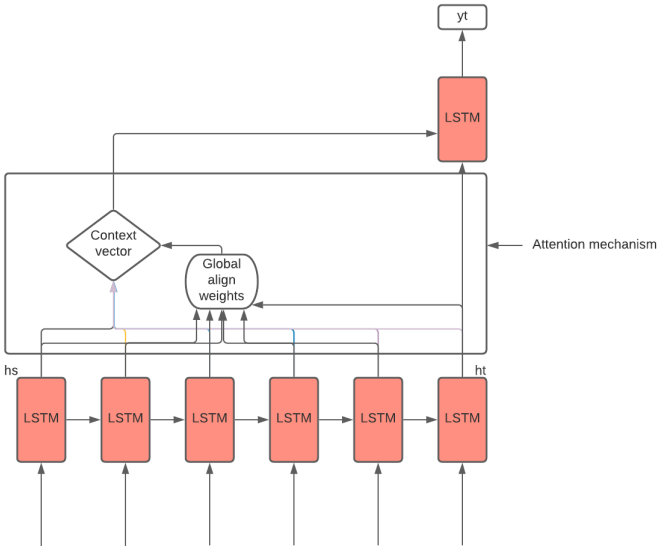

Fig.7 Global attention

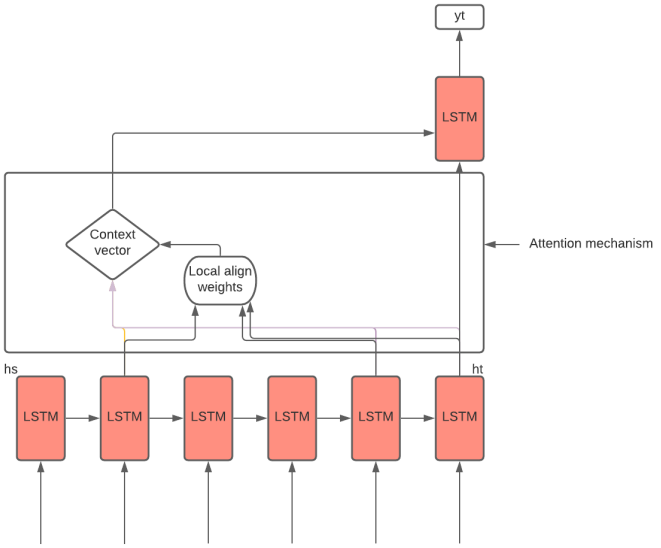

Fig.8 Local Attention

4.2 Complex Design

Even though abstractive text summarizer is better than extractive text summarizer in many aspects, it is way too complicated to design it as compared to the extractive one. Especially if the summarizer is of multiple domains and with good accuracy.

4.3 Time Consumption

Considering the time consumption of cleaning, converting, embedding and vectorizing of the input

sequences and then feeding it into the model, it takes approximately 40 seconds for a low end cpu to generate the summary. It also depends on several other factors such as length of the text, randomness, etc.

4.4 Training obstacles

Since it takes at least 10,000 articles of a single topic to train one model, it is sometimes really hard to get 10,000 genuine articles of one single topic. Many articles turn out to be fake, with grammatical errors, poorly written, generated with poor AI, etc. These uncertain training samples mislead the model and make the model give random words as summary output.

V. FUTURE SCOPE

Summarization of data leads to faster processing of information in many domains such as customer segmentation, fake news analysis, news headline generation, product review analysis, etc. The faster processing and analyzing of people's reviews gives a better insight of the demands of people on a particular product, in turn improving a particular company's performance. Various other applications can be thought of using this particular principle, like sentiment analysis, text translation, fake news detection, spam email classification, etc.

VI. CONCLUSION

The contributions of the analysis and implementation of this project is identified in pursuit of generalizing LSTM model for summarization.

- We explored how different combinations of training data and parameters affected the training performance of the model
- We designed and implemented an existing framework for blind unbiased review that produces more actionable and objective score.

VII. ACKNOWLEDGEMENT

It gives immense pleasure to express our deep sense of gratitude with sincere thanks and appreciation to our project guide Prof. Mukund Kulkarni Sir for suggesting and supporting us to carry the project work.

REFERENCES

[1] Saranyamol C S, Sindhu L, "A Survey on Automatic Text Summarization", International Journal of Computer Science and Information Technologies, 2014,Vol. 5 Issue 6.

[2] Vishal Gupta,"A Survey of Recent Keywords and Topic Extraction Systems for Indian Languages", International Journal of Engineering Trends and Technology (IJETT), 2013, Vol. 6 No. 6

[3] Viswanath Meghana, "Thesis: Ontology-Based Automatic Text Summarization", M. Sc Thesis, Vishweshwaraiah Institute of Technology, India, 2009.

[4] Fachrurrozi M., Yusliani Novi, and Yoanita Rizky Utami, "Frequent Term based Text Summarization for Bahasa Indonesia", International Conference on Innovations in Engineering and Technology Bangkok (Thailand), 2013.

[5] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In Text Summarization Branches Out. Association for Computational Linguistics, Barcelona, Spain, 74–81. https://www.aclweb.org/anthology/W04-1013

[6] Ming Zhong, Pengfei Liu, Yiran Chen, Danqing Wang, Xipeng Qiu, and X. Huang. 2020. Extractive Summarization as Text Matching. ArXiv abs/2004.08795 (2020).

[7] Ragunath R. And Sivaranjani N., "Ontology Based Text Document Summarization System Using Concept Terms", ARPN Journal Of Engineering And Applied Sciences, 2015, Vol. 10, No. 6.

[8] Plaza Laura, Díaz Alberto and Gervás Pablo, "A semantic graphbased approach to biomedical summarisation", Artificial Intelligence in Medicine 53, 2011.

[9] Ajmal E. B, Rosna P Haroom, "Summarization of Malayalam Document Using Relevance of Sentence", International Journal of Least Research in Engineering and Technology (IJLRET), 2015, Vol. 1, Issue 6.

[10] Deepali K. Gaikwad "A Review Paper on Text Summarization ". International Journal of Advanced Research in Computer and Communication Engineering Vol. 5, Issue 3, March 2016:

[11] Dixit Rucha S., Apte S. S., "Improvement Of Text Summarization Using Fuzzy Logic Based Method", IOSR Journal Of Computer Engineering (IOSRJCE) ISSN: 2278-0661, ISBN: 2278-8727, 2012, Vol. 5, Issue 6.

[12] Mohamed Ahmed A., Rajasekaran Sanguthevar, "Query-Based Summarization Based on Document Graphs", Document Understanding Conferences, NIST, 2006

[13] Naresh Kumar Nagwani ,Dr. Shrish Verma Associate "A Frequent Term and Semantic Similarity based Single Document Text Summarization Algorithm" , International Journal of Computer Applications (0975 – 8887) Volume 17– No.2, March 2011