A Project Report On

Abstractive Text Summarizer

By

Shreyans Jain

M. Tech.

Department of Computer Science and Engineering,

School of Science and Engineering,

Bennett University

Date: 7Th May, 2018.

Table of Content

Abstract ……1

Introduction ……..……....2

Deep Learning 3

Recurrent Neural Network (RNN) 4

Long Shot Term Memory (LSTM) Units 5

Encoders and Decoders 6

Word Embedding 8

Setting up Environment for Summarizer Using Python 10

Related Work 11

Evaluation Measure 12

Methodology 16

Conclusion 24

References 25

Abstract

Neural sequence-to-sequence models have provided a viable innovative approach for abstractive text summarization (meaning they are not restricted to simply selecting and rearranging passages from the original text). However, these models have two shortcomings: they are liable to reproduce factual details inaccurately, and they tend to repeat themselves [1]. In this work we have used standard Long Short-Term Memory(LSTM) sequence-to-sequence attentional model. This method utilizes a local attention model for generating each word of the summary conditioned on the input sentence. While the model is structurally simple, it can easily be trained end-to-end and scales to a large amount of training data. We apply our model to the Amazon-fine-food-review dataset. We evaluate the reconstructed paragraph using standard metrics like ROUGE, showing that neural models can encode texts in a way that preserve syntactic, semantic, and discourse coherence.

Introduction

With the dramatic growth of the Internet, people are overwhelmed by the tremendous amount of online information and documents. This expanding availability of documents has demanded exhaustive research in automatic text summarization. According to [2] a summary is defined as "a text that is produced from one or more texts, that conveys valuable information in the original text(s), and that is no longer than half of the original text(s) and usually, significantly less than that".

Generating a condensed version of a passage while preserving its meaning is known as text summarization. Tackling this task is a crucial step towards natural language understanding. Summarization systems can be broadly classified into two categories. Extractive models generate summaries by cropping important segments from the original text and putting them together to form a coherent summary. Abstractive models generate summaries from scratch without being constrained to reuse phrases from the original text.

Automatic text summarization is very challenging, because when we as humans summarize a piece of text, we usually read it entirely to develop our understanding, and then write a summary highlighting its main points. Since computers lack human knowledge and language capability, it makes automatic text summarization a very difficult and non-trivial task.

Summaries can also be of two types: generic or query-focused [3][4]. Topic-focused or user-focused summaries are the other names for query-focused summaries. Such a summary includes the query related content whereas a general sense of the information present in the document is provided in a generic summary.

Summarization task can be either supervised or unsupervised. Training data is needed in a supervised system for selecting important content from the documents. Large amount of labelled or annotated data is needed for learning techniques. These systems are addressed at sentence level as two-class classification problem in which sentences belonging to the summary are termed as positive samples and sentences not present in the summary are named as negative samples. For performing sentence classification, some popular classification methods are employed such as Support Vector Machine (SVM) [4] and neural networks [5]. On the other hand, unsupervised systems do not require any training data. They generate the summary by accessing only the target documents. Thus, they are suitable for any newly observed data without any advanced modifications. Such systems apply heuristic rules to extract highly relevant sentences and generate a summary. The technique employed in unsupervised system is clustering.

Based on the style of output, there are two types of summaries: indicative and informative summaries. Indicative summaries tell what the document is about. They give information about the topic of the document. Informative summaries, while covering the topics, give the whole information in elaborated form.

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.

Before we move on to further things lets discuss about the basic concepts, so that it will be easy to move ahead. Concepts that are covered are, Deep Learning, Recurrent Neural Network(RNN), Long Short Term Memory(LSTM) Cells, Encoders-Decoders, Attention Mechanism and Word Embedding.

➢ Deep Learning

In this project we are going to use the concept of Deep Learning for abstractive summarizer based on food review dataset. So before developing the model, lets understand the concept of deep learning. The basic structures of neural network with its hidden layer is shown in the following figure 1.

Neural Networks (NN) are also used for Natural Language Processing (NLP), including Summarizers. Neural networks are effective in solving almost any machine learning classification problem. Important parameters required in defining the architecture of neural network (NN) are number of hidden layers to be used, number of hidden units to be present in each layer, activation function for each node, error threshold for the data, the type of interconnections, etc. neural networks can capture very complex characteristics of data without any significant involvement of manual labour as opposed to the machine learning systems. Deep learning uses deep neural networks to learn good representations of the input data, which can then be used to perform specific tasks.

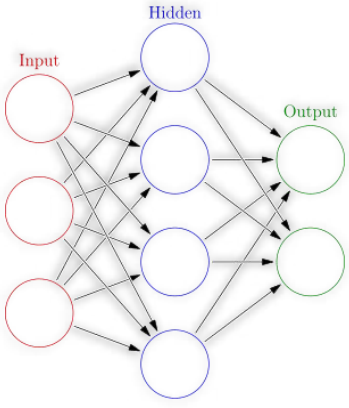


Figure 1: Basic Structure of NN

➢ Recurrent Neural Network (RNN)

Recurrent Neural Networks were created in the 1980's but have just been recently gaining popularity from advances to the networks designs and increased computational power from graphic processing units. They're especially useful with sequential data because each neuron or unit can use its internal memory to maintain information about the previous input. This is great because in cases of language, "I had washed my house" is much more different than "I had my house washed". This allows the network to gain a deeper understanding of the statement.

This is important to note because reading through a sentence even as a human, you're picking up the context of each word from the words before it. A RNN has loops in them that allow infromation to be carried across neurons while reading in input.

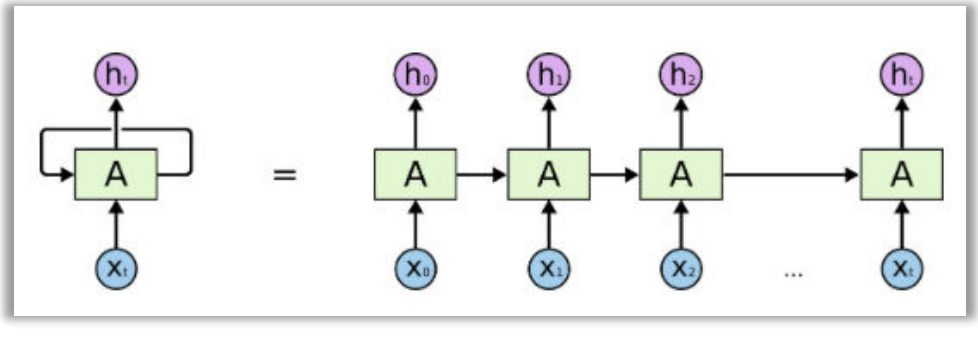


Figure 2: Unrolled RNN Unit

In these diagrams x_t is some input, A is a part of the RNN and h_t is the output. Essentially you can feed in words from the sentence or even characters from a string as x_t and through the RNN it will come up with a h_t.

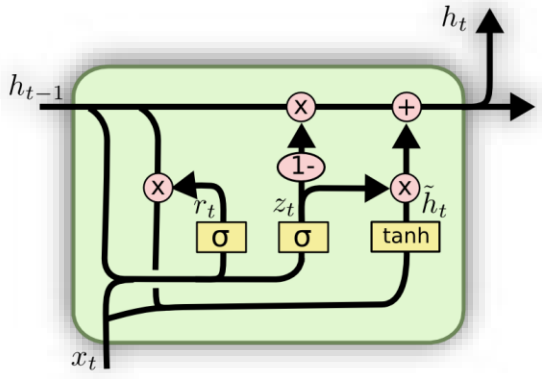
The goal is to use h_t as output and compare it to your test data (which is usually a small subset of the original data). You will then get your error rate. After comparing your output to your test data, with error rate in hand, you can use a technique called Back Propagation Through Time (BPTT). BPTT back checks through the network and adjusts the weights based on your error rate. This adjusts the network and makes it learn to do better.

Theoretically RNNs can handle context from the beginning of the sentence which will allow more accurate predictions of a word at the end of a sentence. In practice this isn't necessarily true for vanilla RNNs. This is a major reason why RNNs faded out from practice for a while until some great results were achieved with using a Long Short Term Memory(LSTM) unit inside the Neural Network.

➢ Long Short Term Memory (LSTM) Units

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.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 3: Architecture of LSTM Unit

➢ Encoders 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.

Since our approach involves implementation of Bi-directional Encoders, we will discuss little more about these. The following figure 4 shows the structure of Bi-directional LSTM's.

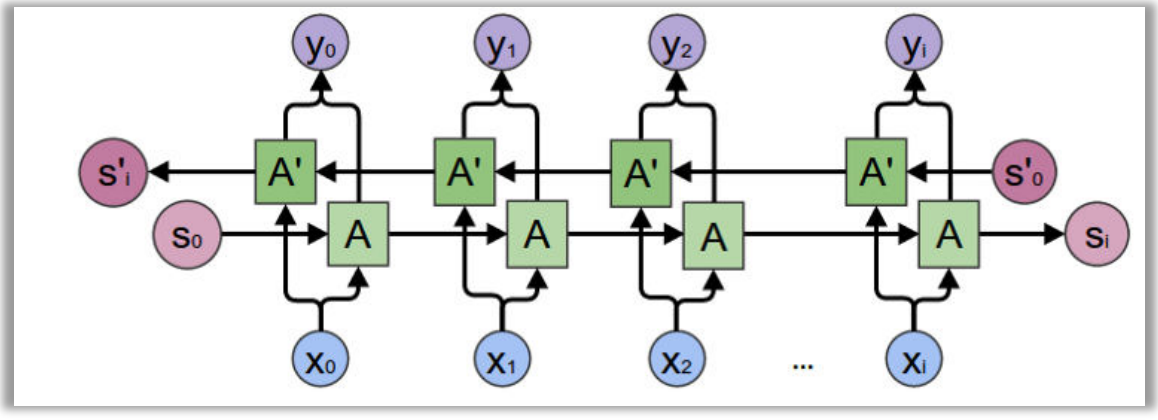


Figure 4: Bi-directional LSTM's

Bidirectional recurrent neural networks(RNN) are just putting two independent RNNs together. The input sequence is fed in normal time order for one network, and in reverse time order for another. The outputs of the two networks are usually concatenated at each time step, though there are other options, e.g. summation. This structure allows the networks to have both backward and forward information about the sequence at every time step. The concept seems easy enough. But when it comes to implementing a neural network which utilizes bidirectional structure, confusion arises…

The Confusion:

1. The way to forward the outputs of a bidirectional RNN to a dense neural network. For normal RNNs we could just forward the outputs at the last time step, and the following picture I found via Google shows similar technique on a bidirectional RNN. But, if we pick the output at the last time step, the reverse RNN will have only seen the last input (X_3 in the picture). It'll hardly provide any predictive power.

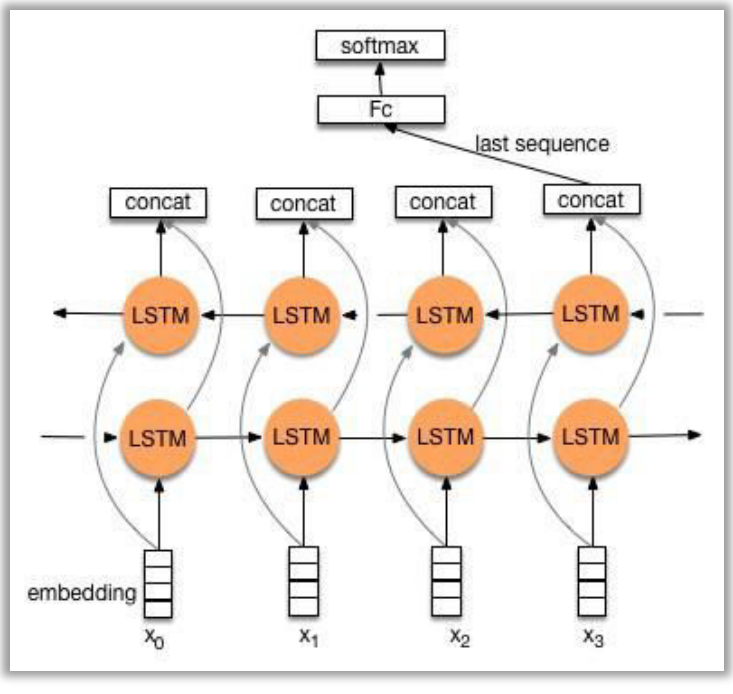


Figure 5: A Confusion Formulation.

2. The second confusion is about the returned hidden states. In seq2seq models, we'll want hidden states from the encoder to initialize the hidden states of the decoder. Intuitively, if

we can only choose hidden states at one-time step, we'd want the one at which the RNN just consumed the last input in the sequence. But if the hidden states of time step n (the last one) are returned, as before, we'll have the hidden states of the reversed RNN with only one step of inputs seen.

➢ Word Embedding

Before diving deep in what are word embeddings? Why it is used? Etc. Let's 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)

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. It is built using an ensemble that combines data from ConceptNet, word2vec, GloVe, and OpenSubtitles 2016, using a variation on retrofitting. It is described in the paper ConceptNet 5.5: An Open Multilingual Graph of General Knowledge, presented at AAAI 2017.

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 3.5 and above (Anaconda 3 installation is recommended).
- Download and install NumPy
- Download and install Pandas
- Download and install NLTK Library
- Download and install TensorFlow v1.1
- Make sure whether Jupyter Notebook is pre-installed, as we will be using Jupyter Notebook.
- Download the dataset from here.

Related Work

Ganesan et al. (2010) [6] presented Opinosis, a new summarization approach that makes use of graphs for generating concise abstractive summaries of highly redundant opinions. Opinosis is highly flexible as it does not require any domain knowledge and it uses shallow NLP. In this approach, firstly a textual graph is made, representing the text to be summarized. Then, for generating candidate abstractive summaries, various sub-paths in the graph are explored and scored by making use of three unique properties of graphs. Evaluation results conclude that summaries generated by Opinosis have reasonable agreement with human summaries. Moreover, readable, concise, well-formed and informative summaries are generated that contain important content. This system is evaluated on reviews of hotels, cars and various products and obtains scores for ROUGE-1 recall as 0.2831, ROUGE-2 recall as 0.0853 and ROUGE-SU4 recall as 0.0851.

Kallimani et al. (2011) [7] implemented various statistical approaches for abstractive summarization of Telugu documents. The proposed system pre-processes, summarizes and post-processes each document. For summarization, many key features are utilized to generate a summary like word clues, keyword extraction, sentence selection, sentence extraction and summary generation. Finally, during post-processing, extractive summary is converted to abstractive summary by employing summary refinement and summary rephrasing. The precision obtained for keyword selection over a set of samples is 70%.

Lloret and Palomar (2011a, b) [8][9] proposed an approach for abstractive text summarization by employing word graphs. This approach compresses and merges information from sentences to form new sentences. Then, an extractive text summarization approach, COMPENDIUM is utilized for determining which of the novel sentences should be selected for forming an abstractive summary. Different approaches are analysed to discuss issues related to generation of abstracts like how to generate new sentences, order in which relevant content can be selected and length of the sentences. Results show that generation of abstracts is a challenging task. However, experiments prove that by combining extractive and abstractive information, abstracts of better quality can be obtained. ROUGE score of 0.405 is obtained on DUC 2002 by combining both extractive and abstractive approaches.

Genest and Lapalme (2011) [10] proposed an approach based on the concept of Information Items (INIT) which is the smallest element of coherent information in the text or a sentence. It can be as simple as an entity's characteristic or as complex as the complete description of an event. This approach has four operational steps, i.e. INIT retrieval, sentence generation, sentence selection and summary generation. This

approach tries to control the content and structure of the document. Evaluation results on the dataset of TAC 2010 are quite satisfactory. This abstraction system generates summary with pyramid score of 0.315, linguistic quality as 2.174 and overall responsiveness as 2.304

Moawad and Aref (2012) [11] proposed a new method for generating an abstract for single document through a reduction method based on semantic graph. This approach works in three phases: firstly, generation of a rich semantic graph from the original documents, secondly reduction of the rich semantic graph thus generated to a highly abstracted graph and finally generation of an abstract. It has been shown by a simulated case study that the given technique minimizes the original text to 50%

Lloret and Palomar (2013) [12] proposed a text summarizer, COMPENDIUM, that creates abstracts of biomedical papers. There are two variants of COMPENDIUM, COMPENDIUME for generating extracts and COMPENDIUME-A that contains both abstractive and extractive methods in which after choosing important sentences, information compression and fusion stage is implemented. Then qualitative and quantitative evaluation of this system was done in which it was concluded that COMPENDIUM is suitable for generating summaries as both of its variants can select important content from the source document but abstractive-oriented summaries produced by COMPENDIUME-A are more appropriate from a human perspective. For specialized journal of medicine, ROUGE-1 score for COMPENDIUME is 44.02%whereas for COMPENDIUME-A is 38.66%.

Khan et al. (2015) [13] proposed an abstractive approach in which summary is not generated by simply selecting sentences from source documents but by semantic representation of the source documents. In this approach, SRL is employed to represent the content of the source document through predicate argument structures. These semantically similar predicate argument structures are clustered by employing semantic similarity measure and then these structures are ranked based on features weighted and optimized by Genetic Algorithm. Experimental results prove that the given approach performs better than the other comparison models and it stood second to the average of human model summaries. On DUC 2002, this abstractive approach has a pyramid score (mean coverage score) of 0.50 and average precision of 0.70.

Banerjee et al. (2015) [14] develops an abstractive summarizer which initially selects the most important document from the multi-document set. Then each sentence from the most important document is used to generate separate clusters. Sentences of other documents that have highest similarity with the cluster sentence are assigned to that cluster. Through a word-graph structure formed from the sentences of each cluster, K-shortest paths are generated. Finally, sentences are selected from the set of shortest paths by

employing a novel integer linear programming (ILP) problem that maximizes information content and linguistic quality and reducing redundancy in the final summary. Experimental evaluation on DUC 2004 and DUC 2005 datasets show that the ROUGE scores of the proposed system are better than the best extractive summarizer on both the datasets and this system outperforms an abstractive summarizer based on multi-sentence compression. On DUC 2004, ROUGE-2 score is 0.11992 and ROUGE-SU4 score is 0.14765 and on DUC 2005 ROUGE-L score is 0.35772 and ROUGE-SU4 score is 0.12411

Bing et al. (2015) [15] proposed an abstraction-based summarization system for multiple documents that create new sentences by exploring more fine-grained syntactic units like noun or verb phrases than sentences. Initially a pool of concepts and facts, represented by noun or verb phrases is extracted from the input documents. Then, a salience score is calculated for each phrase by exploiting redundancy of the document content. To achieve a global optimum solution, phrases are selected and merged simultaneously leading to the creation of new sentences whose validity is ensured through an integer linear optimization model. Experimental evaluations are carried out on TAC 2011dataset using an automated pyramid evaluation metric. The proposed system scores 0.905 and 0.793 at thresholds 0.6 and 0.65 respectively which is better than the other systems in TAC 2011. Also, this system outperforms the other systems on manual linguistic quality evaluation

Evaluation Measure

Evaluation of summary is a very important task in the field of automatic summarization of text. Evaluating the summary besides enhancing development of reusable resources and infrastructure helps in comparing and replicating results and thus, adds competition to improve the results. However, it is practically impossible to manually evaluate multiple documents for obtaining an unbiased view. Therefore, reliable automatic evaluation metrics are required for fast and consistent evaluation. Evaluation of summary is a challenging work too as it is not easy for humans to know what kind of information should be present in the summary. Information changes according to the purpose of the summary and to capture this information automatically, is a challenging task. Figure 1 below describes the taxonomy of summary evaluation measures. Following are the two ways for determining the performance of text summarization:

- Extrinsic evaluation: It determines summary's quality based on how it affects other tasks (Text classification, Information retrieval, Question answering), i.e., a summary is termed as a good summary if it provides help to other tasks. Various methods for extrinsic evaluation are:
    - Relevance assessment: Here various methods are used for evaluating a topic's relevance present in the summary or the original document.
    - Reading comprehension: It determines whether it can answer multiple choice tests after reading the summary.
- Intrinsic evaluation: It determines the summary quality based on coverage between the machine-made summary and the human-made summary. Quality or in formativeness are the two important aspects based on which a summary is evaluated. Usually the in formativeness of a summary is evaluated by comparing it with a human-made summary, i.e., reference summary. There is another paradigm too, i.e. fidelity to the source which checks whether the summary consists of the same or similar content as present in the original document. There is a problem with this method, i.e. how to know which concepts in the document are relevant and which are not.

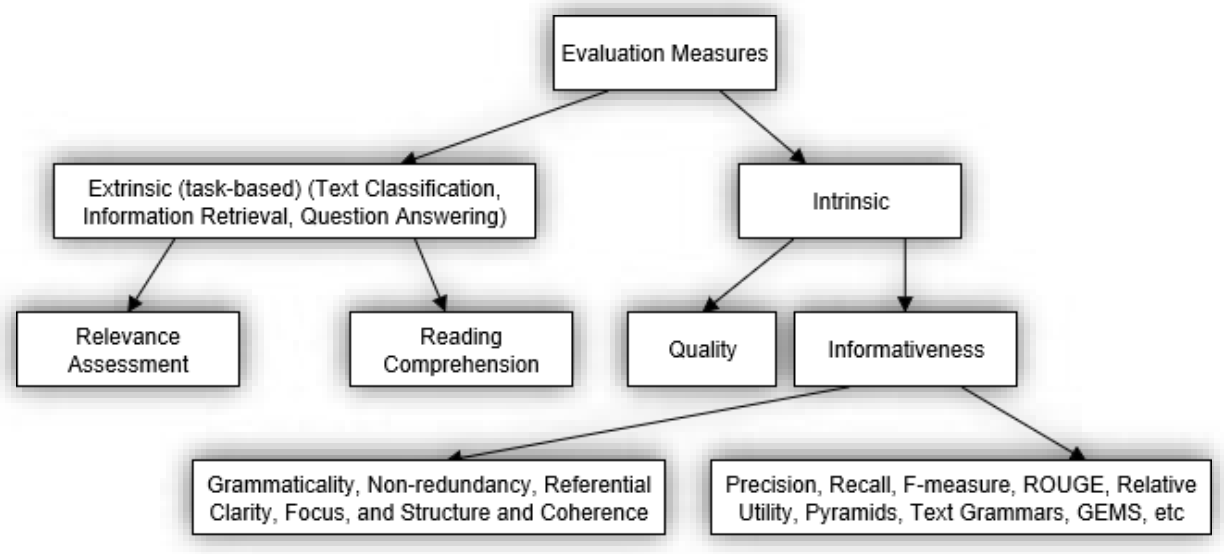


Figure 6: Evaluation Measures [16]

Methodology

The proposed approach uses machine and deep learning concepts. The flow chart for this approach is as follows:

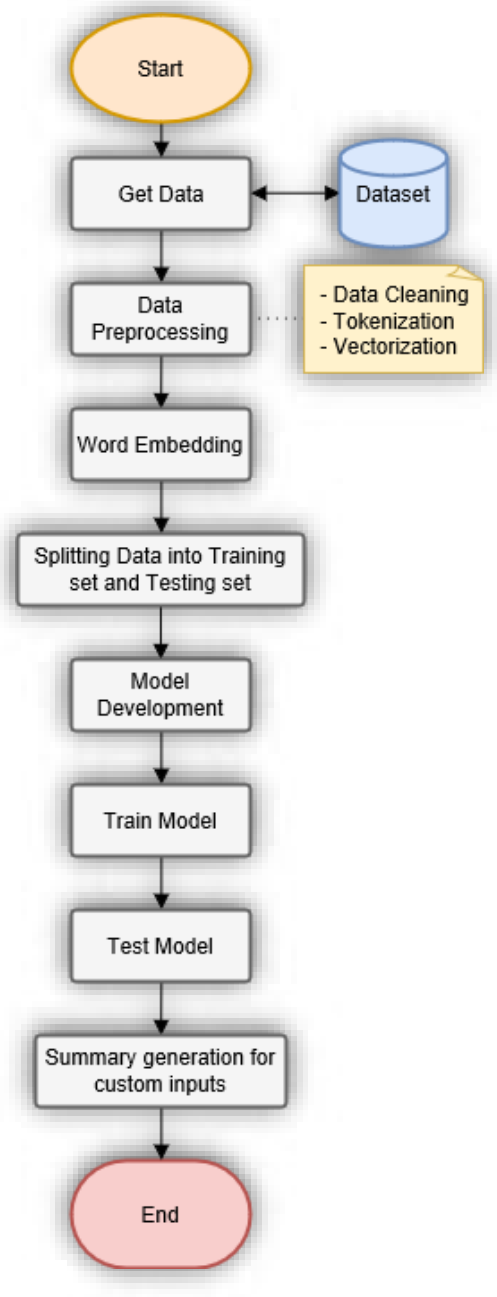


Figure 7: Flow Chart of Project Approach

The above figure 7 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 up to October 2012. 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 8.

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1.0	1.0	5.0	1.303862e+09	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0.0	0.0	1.0	1.346976e+09	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1.0	1.0	4.0	1.219018e+09	"Delight" says it all	This is a confection that has been around a fe...
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3.0	3.0	2.0	1.307923e+09	Cough Medicine	If you are looking for the secret ingredient i...
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0.0	0.0	5.0	1.350778e+09	Great taffy	Great taffy at a great price. There was a wid...

Figure 8: Preview of 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 processed 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

Text:  confection around centuries light pillowy citrus gelatin nuts case filberts cut tiny squares liberally coated powdered s
ugar tiny mouthful heaven chewy flavorful highly recommend yummy treat familiar story c lewis lion witch wardrobe treat seduces
edmund selling brother sisters witch
Summaries:   delight  says it all

Text:  looking secret ingredient robitussin believe found got addition root beer extract ordered good made cherry soda flavor m
edicinal
Summaries:  cough medicine

Text:  great taffy great price wide assortment yummy taffy delivery quick taffy lover deal
Summaries:  great taffy
```

Figure 9: After Cleaning the Data

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 10.

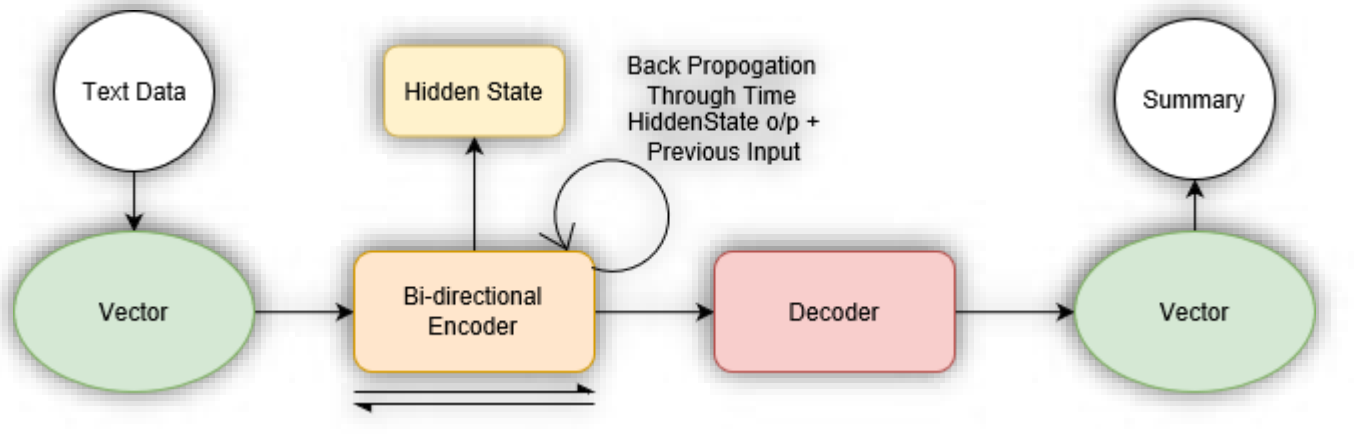


Figure 10: Architecture of Model

Attentive Recurrent Architecture

While designing Decoder we will implement Bahdanau Attention. Let x denote the input sentence consisting of a sequence of M words $x = [x_1, \ldots, x_M]$, where each word x_i is part of vocabulary V, of size $|V| = V$. Our task is to generate a target sequence $y = [y_1, \ldots, y_N]$, 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: $P(y|x) = P(y|x; \theta)$. 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:

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

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

Recurrent Decoder

The above conditional probablity can be modeled using RNN:

$P(y_t|\{y_1, ... , y_{t-1}\}, x; \theta) = P_t = g_{\theta 1}(h_t, c_t)$

Where h_t is the hidden state of the RNN: $h_t = g_{\theta 1}(y_{t-1}, h_{t-1}, c_t)$

Here c_t is the output of the encoder module (detailed in §3.2). It can be a context vector which is computed as a function of the current state h_{t-1} and the input sequence x.

Our Elman RNN takes the following form (El- man, 1990):

$h_t = \sigma(W_1 y_{t-1} + W_2 h_{t-1} + W_3 c_t)$

$P_t = \rho(W_4 h_t + W_5 c_t)$,

where σ is the sigmoid function and ρ is the soft- max, defined as:

$\rho(O_t) = e^{Ot} / \sum_j e^{Oj}$ and W_i $(i=1,..,5)$ are matrices of learnable parameters of sizes $W_{\{1,2,3\}} \in R^{d \times d}$ and $W_{\{4,5\}} \in R^{d \times V}$.

The LSTM decoder is defined as

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

Operator Θ refers to component-wise multiplication, and W_i $(i = 1, . . ., 14)$ are matrices of learnable parameters of sizes $W_{\{1,...,12\}} \in R^{d \times d}$, and $W_{\{13,14\}} \in R^{d \times V}$.

Attentive Encoder

We now give the details of the encoder which computes the context vector c_t for every time step t of the decoder above. With a slight overload of notation, for an input sentence x we denote by x_i the d dimensional learnable embedding of the i[th] word $(x_i \in R^d)$. In addition the position i of the word x_i is also associated with a learnable embedding l_i of size d $(l_i \in R^d)$. Then the full embedding for i[th] word in x is given by $a_i = x_i + l_i$. Let us denote by $B_k \in R^{q \times d}$ a learnable weight matrix which is used to convolve over the full embeddings of consecutive words. Let there be d such matrices $(k \in \{1, \ldots, d\})$. The output of convolution is given by:

$$z_{ik} = \sum_{h=-q/2}^{q/2} a_{i+h} \cdot b^k_{q/2+h},$$

where b^k_j is the j^{th} column of the matrix B^k. Thus the d dimensional aggregate embedding vector z_i is defined as $z_i = [z_{i1}, \ldots, z_{id}]$. Note that each word x_i in the input sequence is associated with one aggregate embedding vector z_i. The vectors z_i can be a representation of the word which captures the position in which it occurs in the sentence and the context in which it appears in the sentence. In our experiments the width q of the convolution matrix B^k was set to 5. To account for words at the boundaries of x we first pad the sequence on both sides with dummy words before computing the aggregate vectors z_i's. Given these aggregate vectors of words, we compute the context vector c_t (the encoder output) as:

$$c_t = \sum_{j=1}^{M} \alpha_{j,t-1} x_j,$$

Where the weights $\alpha_{j,t-1}$ are computed as

$$\alpha_{j,t-1} = \frac{\exp(z_j \cdot h_{t-1})}{\sum_{i=1}^{M} \exp(z_i \cdot h_{t-1})}.$$

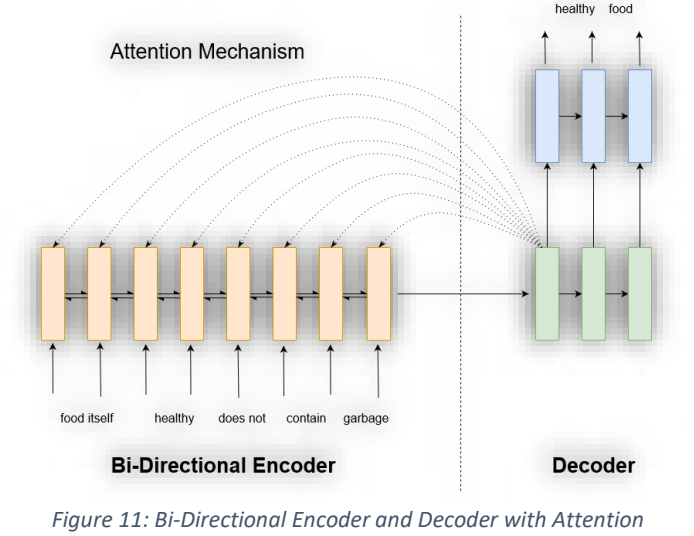


Figure 11: Bi-Directional Encoder and Decoder with Attention Mechanism

Training Snippet

```
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
Epoch  19/50 Batch  465/485 - Loss:  1.574, Seconds: 17.88
Epoch  19/50 Batch  470/485 - Loss:  1.592, Seconds: 18.70
Epoch  19/50 Batch  475/485 - Loss:  1.600, Seconds: 20.63
Epoch  19/50 Batch  480/485 - Loss:  1.644, Seconds: 19.97
Average loss for this update: 1.365
No Improvement.
Epoch  20/50 Batch    5/485 - Loss:  1.573, Seconds: 6.71
Epoch  20/50 Batch   10/485 - Loss:  0.937, Seconds: 7.41
Epoch  20/50 Batch   15/485 - Loss:  1.004, Seconds: 6.14
Epoch  20/50 Batch   20/485 - Loss:  0.965, Seconds: 8.00
Epoch  20/50 Batch   25/485 - Loss:  1.013, Seconds: 5.69
Epoch  20/50 Batch   30/485 - Loss:  0.960, Seconds: 8.01
Epoch  20/50 Batch   35/485 - Loss:  1.011, Seconds: 7.54
Epoch  20/50 Batch   40/485 - Loss:  1.076, Seconds: 5.72
Epoch  20/50 Batch   45/485 - Loss:  1.029, Seconds: 7.38
Epoch  20/50 Batch   50/485 - Loss:  1.026, Seconds: 8.13
Epoch  20/50 Batch   55/485 - Loss:  0.976, Seconds: 5.79
Epoch  20/50 Batch   60/485 - Loss:  0.971, Seconds: 5.15
Epoch  20/50 Batch   65/485 - Loss:  0.974, Seconds: 6.37
```

```
Epoch  20/50 Batch   70/485 - Loss:  0.921, Seconds: 7.12
Epoch  20/50 Batch   75/485 - Loss:  1.080, Seconds: 7.53
Epoch  20/50 Batch   80/485 - Loss:  0.978, Seconds: 7.57
Epoch  20/50 Batch   85/485 - Loss:  0.961, Seconds: 7.09
Epoch  20/50 Batch   90/485 - Loss:  0.987, Seconds: 6.67
Epoch  20/50 Batch   95/485 - Loss:  0.956, Seconds: 7.16
Epoch  20/50 Batch  100/485 - Loss:  0.921, Seconds: 9.17
Epoch  20/50 Batch  105/485 - Loss:  0.947, Seconds: 9.66
Epoch  20/50 Batch  110/485 - Loss:  0.929, Seconds: 8.53
Epoch  20/50 Batch  115/485 - Loss:  1.019, Seconds: 9.14
Epoch  20/50 Batch  120/485 - Loss:  0.918, Seconds: 8.39
Epoch  20/50 Batch  125/485 - Loss:  1.038, Seconds: 7.92
Epoch  20/50 Batch  130/485 - Loss:  0.969, Seconds: 6.86
Epoch  20/50 Batch  135/485 - Loss:  0.992, Seconds: 8.53
Epoch  20/50 Batch  140/485 - Loss:  1.016, Seconds: 7.55
Epoch  20/50 Batch  145/485 - Loss:  0.998, Seconds: 8.28
Epoch  20/50 Batch  150/485 - Loss:  0.947, Seconds: 7.65
Epoch  20/50 Batch  155/485 - Loss:  1.010, Seconds: 8.14
Epoch  20/50 Batch  160/485 - Loss:  0.970, Seconds: 6.96
Average loss for this update: 1.002
No Improvement.
Stopping Training.
Model Trained
```

Loss Graph

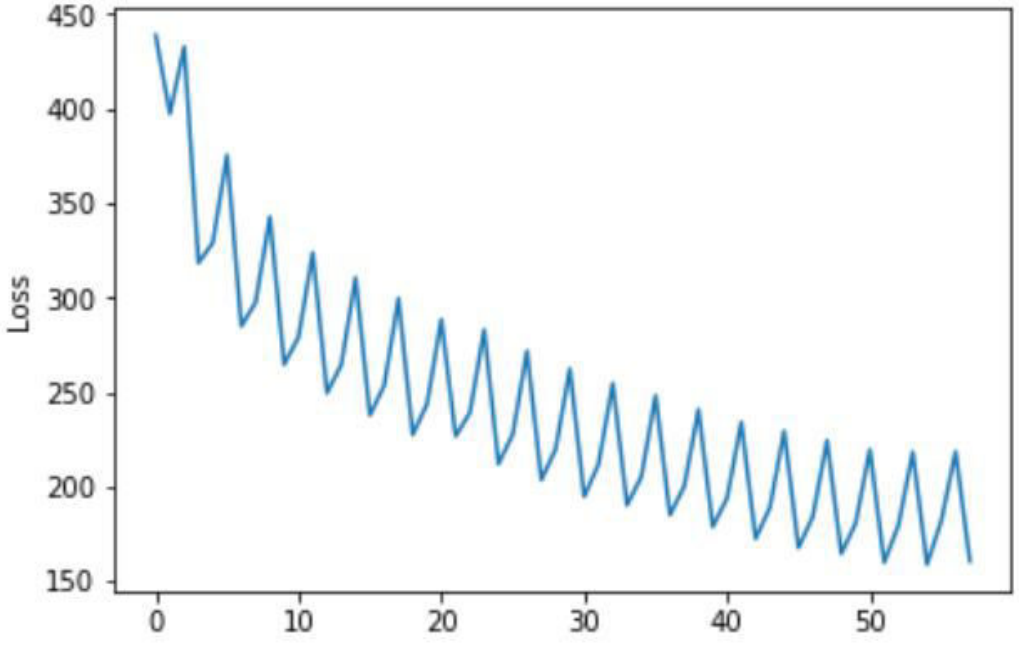


Figure 12: Loss at every iteration

Testing Snippet

Cleaned Text [113, 626, 1687, 68, 35, 1644, 1519, 1739, 0, 800, 2162, 1519, 1739, 1375, 24, 471]
INFO:tensorflow:Restoring parameters from ./best_model.ckpt
Index Value: 10997

Original Text: I like the slight pineapple flavor in this better than the plain coconut water. Good well chilled. Coconut water is supposed to be a healthy drink.
File Written to text.001.txt

Orignal Summary: Good taste

Text
Word Ids: [113, 626, 1687, 68, 35, 1644, 1519, 1739, 0, 800, 2162, 1519, 1739, 1375, 24, 471]
Input Words: like slight pineapple flavor better plain coconut water good well chilled coconut water supposed healthy drink

Summary
Word Ids: [15, 54]
File Written to text.A.001.txt
Response Words: nice taste
Time: 22.70325779914856

ROUGE Score after 35 test results

Precision is :0.6
Recall is :0.6
F Score is :0.6000009500000042
Sum of ROUGE Score: 12.346860213552002
Average ROUGE Score = 0.3527674346729143
Count: 35

Conclusion

We have successfully implemented state-of-the-art model for abstractive sentence summarization [17] to a recurrent neural network architecture. The model is a simplified version of the encoder-decoder framework for machine translation [18]. 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.

References

[1] Abigail See, Peter J. Liu, Christopher D. Manning. 2017. Get To The Point: Summarization with Pointer-Generator Networks. arXiv:1704.04368v2 [cs.CL] 25 Apr 2017.

[2] Dragomir R Radev, Eduard Hovy, and Kathleen McKeown. 2002. Introduction to the special issue on summarization. Computational linguistics 28, 4 (2002), 399–408.

[3] Wan X (2008) Using only cross-document relationships for both generic and topic-focused multi-document summarizations. Inf Retr 11(1):25–49

[4] Ouyang Y, Li W, Li S, Lu Q (2011) Applying regression models to query-focused multi-document summarization. Inf ProcessManag 47:227–237

[5] Fattah MA, Ren F (2009) GA, MR, FFNN, PNN and GMM based models for automatic text summarization. Comput Speech Lang 23:126–144. doi:10.1016/j.csl.2008.04.002

[6] Ganesan K, Zhai C, Han J (2010) Opinosis : a graph-based approach to abstractive summarization of highly redundant opinions. In: Proceedings of the 23rd international conference on computational linguistics, pp 340–348

[7] Kallimani JS, Srinivasa KG, Eswara Reddy B (2011). Information extraction by an abstractive text summarization for an Indian regional language. In: Natural language processing and knowledge engineering (NLP-KE), 2011 7th international conference on IEEE, pp 319–322

[8] Lloret E, PalomarM (2011a). Analysing the use of word graphs for abstractive text summarization. In: IMMM 2011, first international conference, pp 61–66

[9] Lloret E, Palomar M (2011b). Text summarisation in progress: a literature review. Artif Intell Rev 37:1–41. doi:10.1007/s10462-011-9216-z

[10] Genest PE, Lapalme G (2011). Framework for abstractive summarization using text-to-text generation. In: Proceedings of the workshop on monolingual text-to-text generation, Association for Computational Linguistics, pp 64–73

[11] Moawad IF, ArefM(2012) Semantic graph reduction approach for abstractive Text Summarization. In: Pro- ceedings of ICCES 2012, 2012 International Conference on Computer Engineering and Systems, pp 132–138. doi:10.1109/ICCES.2012.6408498

[12] Lloret E, PalomarM(2013) Tackling redundancy in text summarization through different levels of language analysis. Comput Stand Interfaces 35:507–518. doi:10.1016/j.csi.2012.08.001

[13] Khan A, Salim N, Jaya Kumar Y (2015) A framework for multi-document abstractive summarization based on semantic role labelling. Appl Soft Comput 30:737–747. doi:10.1016/j.asoc.2015.01.070

[14] Banerjee S Mitra P, Sugiyama K (2015) Multi-document abstractive summarization using ILP based multi- sentence compression. In: Proceedings of the 24th international joint conference on artificial intelligence (IJCAI 2015), pp 1208–1214

[15] Bing L, Li P, Liao Y, LamW, GuoW, Passonneau RJ (2015) Abstractive multi-document summarization via phrase selection and. arXiv preprint arXiv:1506.01597

[16] Gambhir Mahak, Gupta Vishal. (2016). Recent automatic text summarization techniques: a survey. Artificial Intelligence Review, on 29 March, 2016. DOI 10.1007/s10462-016-9475-9.

[17] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1412–1421, Lisbon, Por- tugal, September. Association for Computational Linguistics.

[18] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Ben- gio. 2014. Neural machine translation by jointly learning to align and translate. CoRR, abs/1409.0473.

Michele