Abstract Text Summarization

Optimizers
Soumyajyoti Das, Prakash Samanta, Arnab Dutta Choudhury
Soumyajyoti : sjd.official.20@gmail.com
Prakash : prakashsamanta365@gmail.com
Arnab : arnabdc2000@gmail.com

June 26, 2022

Abstract

This paper aims to approach perfect Abstractive Text Summarization using Encoder Decoder Architecture coupled with Bahdanau's Attention Mechanism. The Encoder generates a context vector which captures the essence of a collection of sentences. The decoder then uses the context vector along with the words generated by the decoder previously to generate a new word of the summary. The precision of the context vector is very important as it encompasses the whole information about the input. The attention mechanism improves the context vector leading to the generation of more informative summaries. We have measured cosine distance to evaluate the similarity between our generated summary and the original summary.

1 Introduction

The process of producing summaries from the huge sets of information while maintaining the actual context of information is called Text Summarization. The summary should be fluent and concise throughout. There are two types of summarizations - Extractive Summarization and Abstractive Summarization.
In Extractive Summarization, we focus on the vital information from the input sentence and extract that specific sentence to generate a summary. There is no generation of new sentences for summary, they are exactly the same that is present in the original group of input sentences.
On the contrary, Abstract Summarization takes an exact sentence to generate a summary. It focuses on the vital information of the original group of sentences and generates a new set of sentences for the summary. This new sentence might not be present in the original sentence.

In recent years, the volume of textual data has rapidly increased, which has generated a valuable resource for extracting and analysing information. To retrieve useful knowledge within a reasonable time period, this information must be summarised. There are quite a few reasons why we do Abstract Text Summarization such as -
1. To make reading easier
2. To save time
3. To helps memorize information easily
4. To boost the work rate efficiency
We have used Many to Many Sequence Modeling using Recurrent Neural Networks to create models that predict the summary of the reviews. Recurrent Neural networks are preferred for this project as they work on sequential input to capture the dependencies among them, which is useful when extracting the context or meaning of a sentence. The model will be trained and tested on 568,454 rows of the Amazon Reviews Dataset. Using the Attention mechanism we will focus on specific keywords while maintaining the context of our sentence.

2 Literature review

This paper has used an approach to abstractive text summarization based on discourse rules, syntactic constraints, and word graph. Discourse rules and syntactic constraints are used in the process of generating sentences from keywords. Word graph is used in the sentence combination process to represent word relations in the text and to combine several sentences into one.[7]

This paper presents a novel framework that combines sequence-to-sequence neural-based text summarization along with structure and semantic-based methodologies.The pre-processing task is a knowledge-based approach, based on ontological knowledge resources, word sense disambiguation, and named entity recognition, along with content generalization, that transforms ordinary text into a generalized form. A deep learning model of attentive encoder-decoder architecture, which is expanded to enable a coping and coverage mechanism, as well as reinforcement learning and transformer-based architectures, is trained on a generalized version of text-summary pairs, learning to predict summaries in a generalized form. The post-processing task utilizes knowledge resources, word embeddings, word sense disambiguation, and heuristic algorithms based on text similarity methods in order to transform the generalized version of a predicted summary to a final, human-readable form.[1]

Here they approached the problem with an encoder-decoder model with LSTM cells called Sequence-to-sequence (abbrev.Seq2Seq) using OpenNMT. We further explored self-supervised pre-trained models such as BART, T5 and Pegasus on our data. We also briefly investigated the GPT-2 model using OpenAI APIs by training the model with a few-shot learning technique.[8]

All the literature that we have come across has not used any concrete similarity measure to evaluate the quality of the abstract summarization. We came across a paper which used ROGUE score to evaluate the similarity between the Original and the Predicted Summary, but we have shown in the later sections why it may not be a very good idea to base the model correctness on it.

We have implemented the cosine similarity as a measure to compare the contextual similarity between two sentences which may or may not be similar in terms of the words contained in them.

3 Proposed methodology

We have used the Text and Summary column from the Amazon Reviews dataset. Text - A collection of sentences whose summary needs to be found. Summary - A summary which has been provided for a given collection of sentences.

3.1 Data Pre-Processing

We have cleaned the data of null values, removed special characters, converted everything to lowercase, used contraction mapping, removed short words and a modified list of stopwords. A modified list has been used to remove stopwords from the text, as the original list contained negative words, whose removal led to loss in meaningful information.

Separate tokenizers have been built for the texts and summaries which convert the texts to integer sequences.

3.2 Model Building

We have implemented the Encoder Decoder Architecture coupled with the Bahdanau Attention Mechanism.

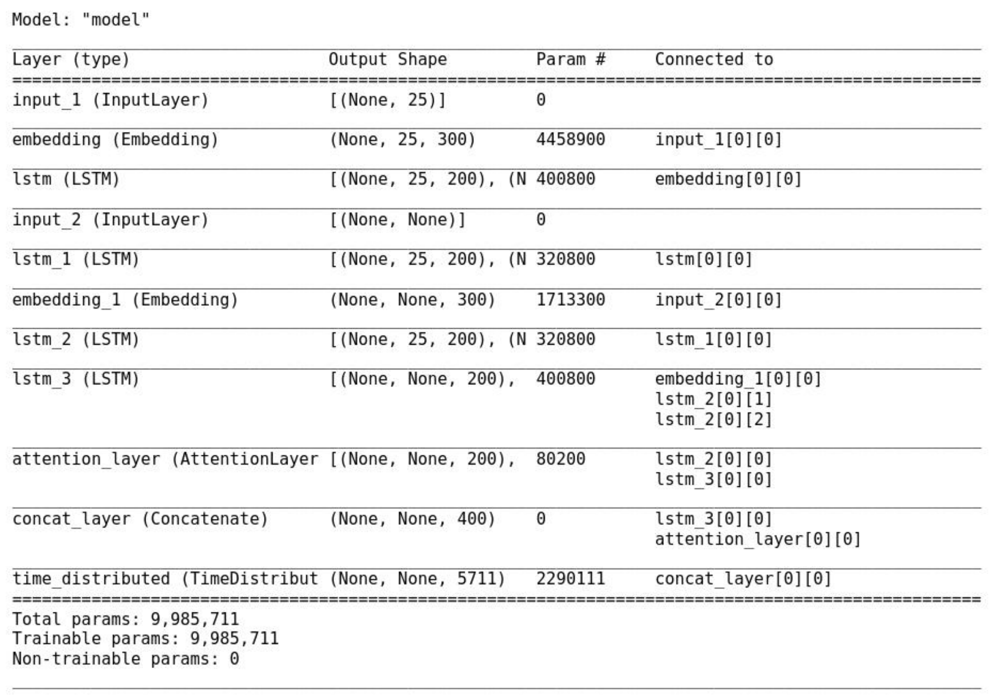
Stacked LSTM has multiple layers of LSTM stacked on top of each other. This leads to a better representation of the sequence.

We have used a 3 layer Stacked LSTM to define our Encoder. A single LSTM has been used to define the Decoder.

The attention mechanism has been implemented between the Encoder and Decoder phases. It takes as an input the outputs of the 3rd layer LSTM and the output states from the Decoder.

The output context is derived by concatenating the result of the Attention Mechanism and the decoder outputs.

A Dense Layer is created as the final layer of the model, with the number of neurons equal to the vocabulary of the summaries, with "softmax" defined as the activation function for this layer.



```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 25)]         0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 25, 300)      4458900     input_1[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                     [(None, 25, 200), (N 400800      embedding[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, 25, 200), (N 320800      lstm[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, None, 300)    1713300     input_2[0][0]
__________________________________________________________________________________________________
lstm_2 (LSTM)                   [(None, 25, 200), (N 320800      lstm_1[0][0]
__________________________________________________________________________________________________
lstm_3 (LSTM)                   [(None, None, 200),  400800      embedding_1[0][0]
                                                                 lstm_2[0][1]
                                                                 lstm_2[0][2]
__________________________________________________________________________________________________
attention_layer (AttentionLayer [(None, None, 200),  80200       lstm_2[0][0]
                                                                 lstm_3[0][0]
__________________________________________________________________________________________________
concat_layer (Concatenate)      (None, None, 400)    0           lstm_3[0][0]
                                                                 attention_layer[0][0]
__________________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 5711)   2290111     concat_layer[0][0]
==================================================================================================
Total params: 9,985,711
Trainable params: 9,985,711
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 1: Model Architecture

Long Short Term Memory(LSTM)

With Recurrent Neural Networks, it cannot capture long-term dependency.In such cases, where the gap between the relevant information and the place that it's needed is small, Recurrent Neural Networks can learn to use the past information. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large. Unfortunately, as that gap grows, Recurrent Neural Networks become unable to learn to connect the information.[4]

This problem can be solved through the implementation of Long Short Term Networks. LSTMs were introduced by Hochreiter and Schmidhuber in 1997.[4]

The key idea of LSTMs is the cell state, or the overall context vector. The context can be thought of to be a memory cell which captures the whole information about the sentence in itself. Through the different words encountered in the sentence, we modify the context to suitably represent the sentence.

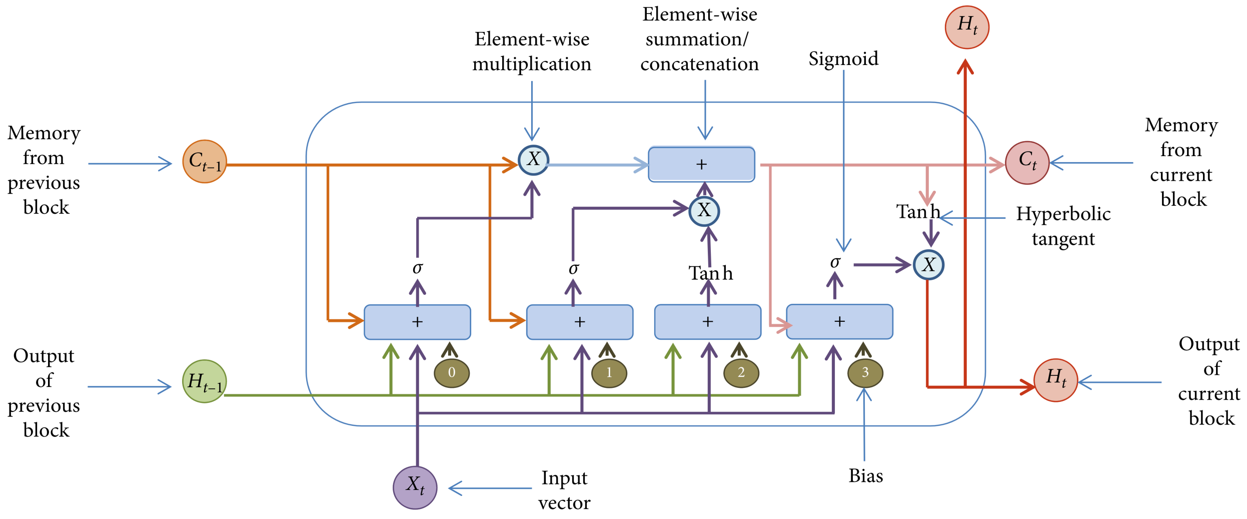


Figure 2: LSTM

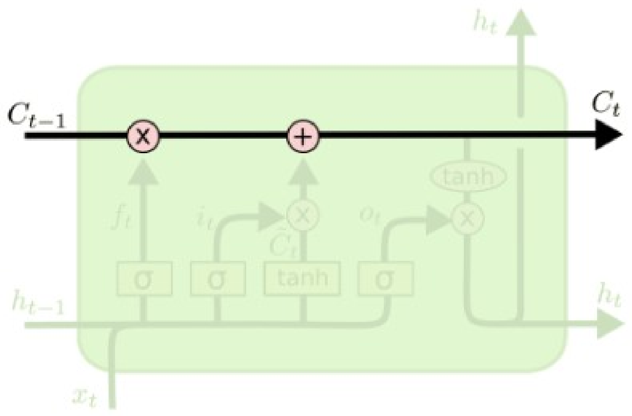
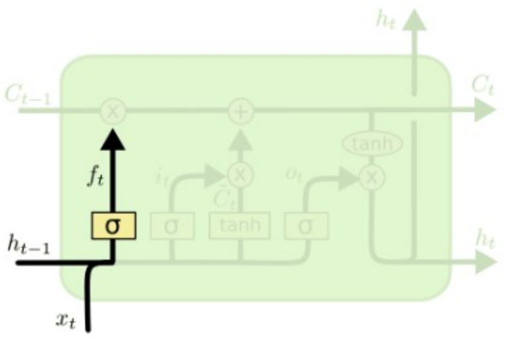


Figure 3: LSTM Context

There are three gates defined which modify the contents of the cell state.[10]

- Forget Gate - Through a combination of the input of the current timestamp, the cell output of the previous timestamp and the sigmoid activation, it decides which portions of the context are irrelevant and should be removed.



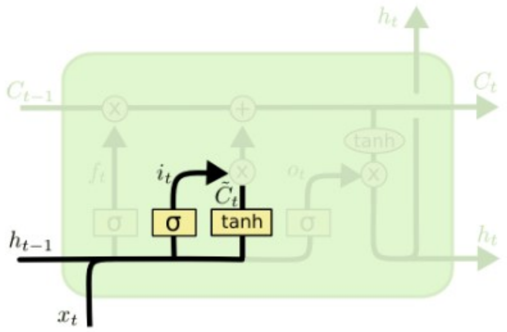
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Figure 4: LSTM Forget Gate

W_f = Weights parameters for the Forget Operation

b_f = Bias values for the Forget Operation

- Input Gate - Through a combination of the input of the current timestamp, the cell output of the previous timestamp and the sigmoid activation, it decides which values of the context to update and similarly through the tanh activation it updates those values.

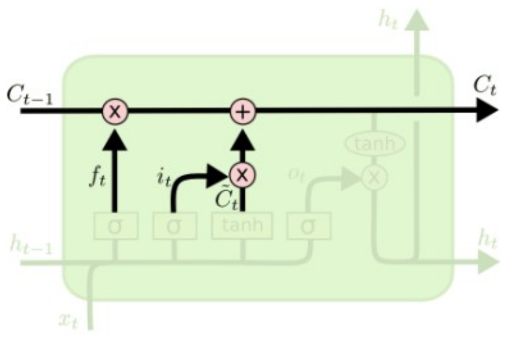


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Figure 5: LSTM Input Gate

W_i = Weights parameters for the Input Operation

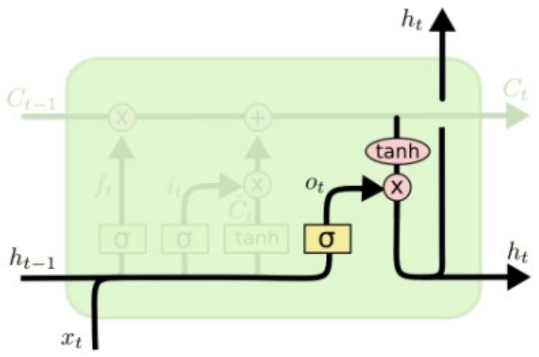
b_i = Bias values for the Input Operation



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 6: LSTM Forget and Input Gate Effect Updating Context

- Output Gate - Through a combination of the input of the current timestamp, the cell output of the previous timestamp and sigmoid activation, it decides the values of the context share as an output of the cell for the current timestep and on multiplication with the tanh activation on the context, it outputs those values.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 7: LSTM Output Gate

W_o = Weights parameters for the Output Operation

b_o = Bias values for the Output Operation

Bahdanau Attention Mechanism

The context vector is generated by the Encoder upon which the Decoder works upon to extract the meaning and consequently generate the summary. If the context vector is defined to be a vector of fixed length then it may not encapsulate the required amount of information to generate a good summary. Thus a fixed length acts as a bottleneck in improving the performance of this basic encoder–decoder architecture.[2] The attention mechanism automatically searches for parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly.

Each time the proposed model generates a word in a translation, it searches for a set of positions in a source sentence where the most relevant information is concentrated. The model then predicts a target word based on the context vectors associated with these source positions and all the previous generated target words.

It encodes the input sentence into a sequence of vectors and chooses a subset of these vectors adaptively while decoding. This allows a model to cope better with long sentences.

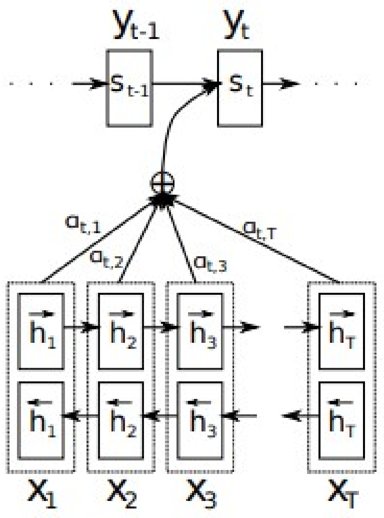


Figure 8: Working of Attention Mechanism

Let $x_1, ..., x_n$ be a sequence of inputs(text words) and $y_1, ..., y_m$ be a sequence of outputs(summary words).

We define each conditional probability of generating a word in the ith timestep as: $p(y_i|y_1, ..., y_{i-1}, x) = g(y_{i-1}, s_i, c_i)$

where s_i is defined as an RNN hidden state for time i computed by, $s_i = f(s_{i-1}, y_{i-1}, c_i)$

Here, unlike the classical Encoder-Decoder architecture, the probability is conditioned on a distinct context vector c_i for each target word y_i

The context vector c_i depends on a sequence of annotations to which an encoder maps the input sequence, $h_1, ..., h_n$

Each annotation contains information about the whole input sequence with a strong focus on the parts surrounding the ith word of the input sequence.

The context vector is then computed as the weighted sum of these annotations.

$$c_i = \sum_{j=1}^{n} a_{ij} h_j$$

Where a_{ij} is the weight of each annotation h_j

The probability a_{ij} reflects the importance of the annotation h_j in generating y_i

Intuitively, this implements a mechanism of attention in the decoder. The decoder decides parts of the source sentence to pay attention to. By letting the decoder have an attention mechanism, we relieve the encoder from the burden of having to encode all information in the source sentence into a fixed length vector. With this new approach the information can be spread throughout the sequence of annotations, which can be selectively retrieved by the decoder accordingly.

3.3 Model Inference

To predict the next word in a summary, the context of the previous timestep and the cell output while predicting the previous word are passed as inputs.

Here, the attention mechanism is found useful. It takes as an input the hidden state output of the decoder and the output state of the decoder.

A Dense Layer is created as the final layer of the model, with the number of neurons equal to the vocabulary of the summaries, with "softmax" defined as the activation function for this layer.

The Dense Layer actually defines the probability that a word in the vocabulary will be the next target word in a summary for a text, given the previous target word in the summary is generated.

The summary keeps on getting generated until the end of summary token is generated or the length of the summary exceeds the maximum summary length we had defined previously while pre-processing the data.

Here the start of summary token's usefulness can be seen in the fact that while predicting the first target word of the summary, we do not have a word previously generated, hence the start token is passed as an input to generate the first word of the summary.

The outputs from the decoder gives us an integer sequence and through reverse indexing of the tokenizer we derive the predicted summary in words.

4 Experimental result

4.1 The Dataset

We have used the Amazon Food Reviews Dataset.[6] The data span a period of more than 10 years, including all 500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories.

- 568,454 reviews
- 256,059 users
- 260 users with greater than 50 reviews

Our project mainly concerns the "Text" and "Summary" columns.

4.2 Experimental Settings

We have defined rare words as words which occur less than 4 times in the whole vocabulary of words. We have also not considered them while generating the embeddings for the encoder.

The texts of word count 25 and less will be fed to the neural network and the summaries of word count 8 and less will be considered for loss calculation. Start and End Tokens have also been added to the summary which will help in the Decoder phase of inference.

Latent Dimension = 200

Embedding Dimension = 300

For the loss function, "Sparse Categorical Cross Entropy" has been used since it converts the integer sequence to a one-hot vector on the fly. This overcomes any memory issues.

The optimizer used is "rmsprop" which takes a convex combination of the square of the gradients in the previous iteration and the gradients in the current iteration.

The concept of early stopping is used on the Validation Loss. It stops the training on the data when the Validation loss increases for two successive iterations. The Validation Set is taken as 10 percent of the total data available for Training. A batch size of 128 is considered.

For calculating the similarity between our generated summaries and existing summaries, we have generated 50000 random indexes from the dataset, calculated the cosine similarity for each of them individually and found their average as the representative similarity for that random set.

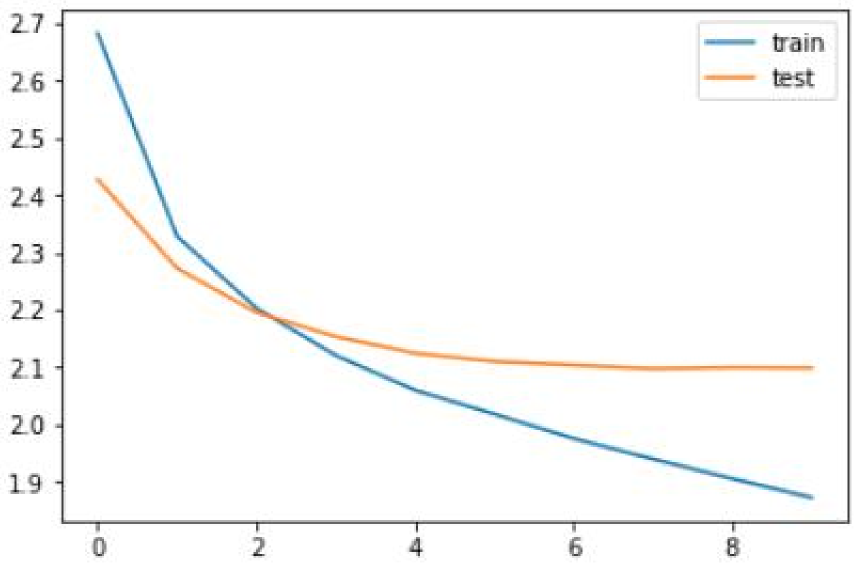


Figure 9: Loss Curve, plotted against no. of Iterations. Notice how because of the Early Stopping Criteria on Validation Loss(denoted by test), it stops in the 10th iteration

4.3 Measure of Accuracy

The procedure employed is Abstractive Text Summarization, since this method generates summaries using words which may or may not be present in the original text, the text matching techniques like ROGUE score and BLEU score are not very effective while calculating the similarity.[11]

They are useful when the generated summary and the original summary both have words in common but Abstractive Text Summarization seeks to capture the context of the sentence and then represent that using the most suitable words. Also when the original summaries contain corrupted or inappropriate summaries, then even if the generated summary is correct it leads to low similarity scores.[5]

We have used the cosine similarity measure that does not compare the exact words of the summaries but instead seeks to compare the general contexts of the sentences.

Cosine Similarity

To compare two sentences(as in our case), using the cosine distance measure. We will have to convert the sentences to vectors.

The cosine distance between two vectors x and y is calculated by: $x.y/||x|| * ||y||$

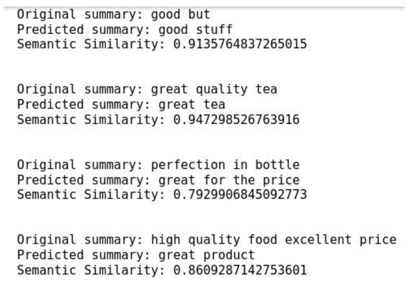
The cosine similarity is measured by $1 - cosinedistance$.

In our project, we have used the pre-trained vector representations of words inferenced from the "glove-wiki-gigaword-50" corpus. Here "50" indicates there are 50 dimensions for vector representation of each word.

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.

Thus, the learned representations are derived depending on the context of the corpus.

Our average cosine semantic similarity over 50000 samples is 0.6913427883907872



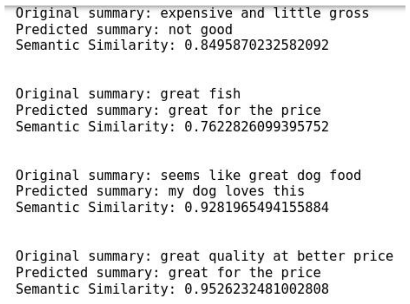
```
Original summary: good but
Predicted summary: good stuff
Semantic Similarity: 0.9135764837265015

Original summary: great quality tea
Predicted summary: great tea
Semantic Similarity: 0.947298526763916

Original summary: perfection in bottle
Predicted summary: great for the price
Semantic Similarity: 0.7929906845092773

Original summary: high quality food excellent price
Predicted summary: great product
Semantic Similarity: 0.8609287142753601
```

Figure 10: Sample Output 1



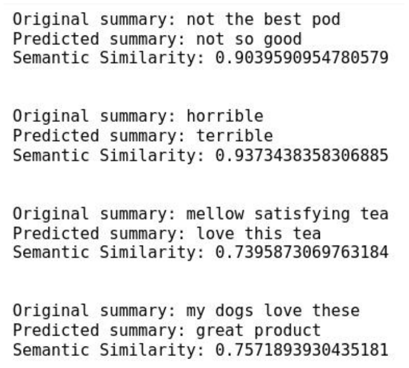
```
Original summary: expensive and little gross
Predicted summary: not good
Semantic Similarity: 0.8495870232582092

Original summary: great fish
Predicted summary: great for the price
Semantic Similarity: 0.7622826099395752

Original summary: seems like great dog food
Predicted summary: my dog loves this
Semantic Similarity: 0.9281965494155884

Original summary: great quality at better price
Predicted summary: great for the price
Semantic Similarity: 0.9526232481002808
```

Figure 11: Sample Output 2



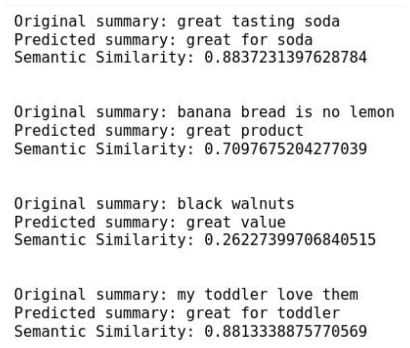
```
Original summary: not the best pod
Predicted summary: not so good
Semantic Similarity: 0.9039590954780579

Original summary: horrible
Predicted summary: terrible
Semantic Similarity: 0.9373438358306885

Original summary: mellow satisfying tea
Predicted summary: love this tea
Semantic Similarity: 0.7395873069763184

Original summary: my dogs love these
Predicted summary: great product
Semantic Similarity: 0.7571893930435181
```

Figure 12: Sample Output 3



```
Original summary: great tasting soda
Predicted summary: great for soda
Semantic Similarity: 0.8837231397628784

Original summary: banana bread is no lemon
Predicted summary: great product
Semantic Similarity: 0.7097675204277039

Original summary: black walnuts
Predicted summary: great value
Semantic Similarity: 0.26227399706840515

Original summary: my toddler love them
Predicted summary: great for toddler
Semantic Similarity: 0.8813338875770569
```

Figure 13: Sample Output 4

5 Summary

We worked with the Amazon Reviews Dataset and and summarised the texts to generate summaries using the Abstractive Text Summarization Technique.

We used the Encoder Decoder Architecture coupled with the Attention Mechanism for the Many to Many Sequence Modelling Problem.

The Encoder was implemented using 3 Stacked LSTM and the Decoder was implemented using a single LSTM.

The cosine similarity measure was used to evaluate the similarities between the original summary and the predicted summary, where the vector representations for the words were imported from the pretrained GLOVE model on glove-wiki-gigaword-50 corpus.

The average cosine similarity for 50000 observations for our model is found to be 0.6913427883907872

References

[1] Andreas Stafylopatis anagiotis Kouris, Georgios Alexandridis. Abstractive text summarization: Enhancing sequence-to-sequence models using word sense disambiguation and semantic content generalization. https://direct.mit.edu/coli/article/47/4/813/106774/Abstractive-Text-Summarization-Enhancing-Sequence, 2021.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Data Flair. Text summarization using machine learning. https://data-flair.training/blogs/machine-learning-text-summarization/, 2019.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[5] Nina Hristozova. To rouge or not to rouge? https://towardsdatascience.com/to-rouge-or-not-to-rouge-6a5f3552ea45#:~:text=When%20the%20aim%20is%20abstractive,factual%20accuracy%20of%20the%20summaries., 2021.

[6] Kaggle. Amazon food reviews. https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews, 2017.

[7] Huong Thanh Le and Tien Manh Le. An approach to abstractive text summarization. In *2013 International Conference on Soft Computing and Pattern Recognition (SoCPaR)*, pages 371–376, 2013.

[8] Gyan Mittal and Dipali Patidar. Abstractive text summarization with deep learning. https://timesinternet.in/blog/abstractive-text-summarization-with-deep-learning/, 2021.

[9] Gyan Mittal and Dipali Patidar. Abstractive text summarization with deep learning. https://timesinternet.in/blog/abstractive-text-summarization-with-deep-learning/, 2021.

[10] Christopher Olah. Understanding lstm networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015.

[11] Stack Overflow. Stack overflow. https://stats.stackexchange.com/questions/451419/rouge-scores-for-extractive-vs-abstractive-text-summarization, 2020.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[13] Analytics Vidhya. Comprehensive guide to text summarization. https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/, 2019.