Text summarization

Rachit Patel

Department of Electrical Engineering and computer science
Loyola Marymount University
rpatel31@lion.lmu.edu

Bryan Moncada

Department of Electrical Engineering and computer science
Loyola Marymount University
bmoncada@lion.lmu.edu

Abstract

Text Summarization has been the most vital task in today's era of the internet. Nowadays, all official documents, profiles, research papers. blogs, news, and customer reviews are accessible through the internet. These data may contain some unnecessary information that the user may not be looking for. In such cases, tasks like information extraction and finding conclusions become so tedious and time-consuming. The text summarization by a deep neural network makes it possible to abbreviate the entire blog or text in a small relevant paragraph that contains essential information. In this document, we have used Amazon fine food reviews as text with its summary to train a sequence-to-sequence model to generate summaries for given food reviews.

Keywords

Text summarization, Deep neural network

1. Introduction

Automatic text summarization can be defined as "As a task of producing a concise and fluent summary while preserving key information content and overall meaning" [1]. Text summarization is often needed while analyzing customers' reviews on E-Commerce websites like Amazon, eBay, Kindle, IMDB, etc. The reviews are usually belabored and needed to be concluded in short notations like ratting or ranking. But it not always possible to convert the entire review text into just a ratting number since a text review may contain some unwanted information that has very little impact on user rating. In such cases, it is required to summarize the entire review in a single-lined summary before applying any data analytics on the reviews.

Text summarization can be divided into two different types

a) Extractive summarization
b) Abstractive summarization

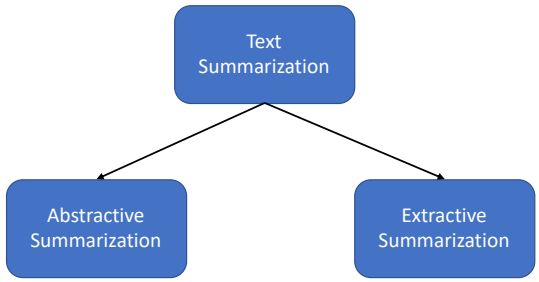


Figure 1.1: Text Summarization

1.1. Extractive Summarization

Extractive summarization techniques produce summaries by choosing a subset of the sentences in the original text [1]. These summaries contain the most important sentences of the input. Input can be a single document or multiple documents.

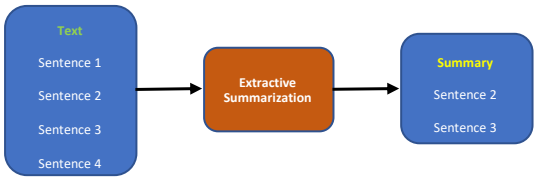


Figure 1.2: Extractive Summarization

1.2. Abstractive summarization

The abstractive summarization approach generates new sentences from the original text. In the contrast to Extractive summarization, the generated sentences

may not be present in the source text. But the generated summary is contextually similar to the source text. This approach of text summarization can produce the same summaries that a human could produce.

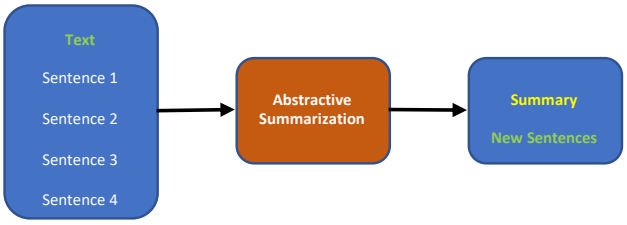


Figure 1.3: Abstractive Summarization

1.3. Sequence to sequence modeling

sequence-to-sequence models constitute a common framework for solving sequential problems [2]. In seq2seq models, the input is a sequence of certain data units and the output is also a sequence of data units. Traditionally, these models are trained using a ground-truth sequence via a mechanism known as teacher forcing, where the teacher is the ground-truth sequence [2]. A standard sequence-to-sequence model architecture contains an Encoder and Decoder. The Encoder-Decoder architecture is mainly used to solve the sequence-to-sequence (Seq2Seq) problems where the input and output sequences are of different lengths[3].

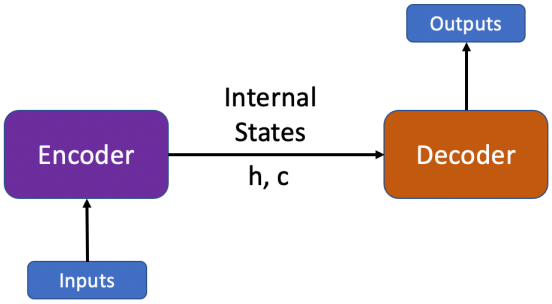


Figure 1.4: Seq2Seq with encoder-decoder

Figure 1.4 shows a basic architecture of the Sequence-to-sequence model, where a review text is faded to the encoder as an input and the summary of that review is produce by the decoder as an output. In this case, the input sequence is significantly larger than the output sequence. In this mechanism, the encoder and the decoder contain deep neural components that to be trained during the training of the seq2seq model. There are three types of Encoder-Decoder components that can be used in the seq2seq model.

a) RNN – Recurrent Neural Network
b) LSTM – Long Short-Term Memory
c) GRU – Gated recurrent units

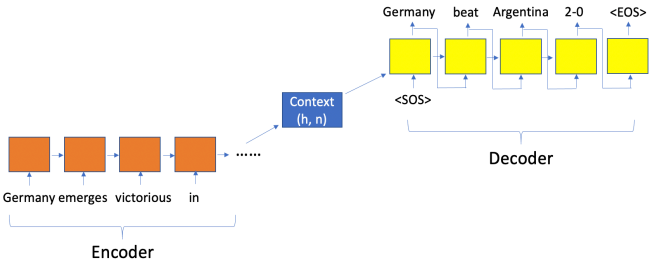


Figure 1.5: Summary generation by Encoder-Decoder

2. Problem Definition

The dataset is the open-source reviews of fine food from amazon. Given a series of input text sequences and output summaries, that task is to build, train, and evaluate the model to produce summaries for some unseen food reviews.

3. Dataset and preprocessing

The dataset of food reviews consists of reviews and their summaries. There are over 700,000 reviews available in the dataset, collected in the span of more than 10 years. The data is available in .csv file format. We have used a sample of 100,000 reviews to reduce the training time of the model.

3.1. Preprocessing

The dataset contains duplicate values, NA values, and some unwanted symbols or characters. This unclear data needs to be cleaned first by performing the following operation.

i. Drop Duplicates and NA values
ii. Drop unwanted symbols and characters
iii. Convert all characters to lowercase
iv. Remove HTML tags
v. Contraction mapping
vi. Remove texts inside the parenthesis
vii. Eliminate punctuation and special characters
viii. Remove stop-words
ix. Remove short words
x. Add <start> and <end> at the beginning and the end of each summary.
xi. Tokenizing

3.2. Data distribution

We used the dataset with different length from 1000 to 100,000. Each time we use 80% of the total data for training and 20% of the total data for validation. We have tested our model on the validation dataset to check the model's performance after removing the overfitting.

4. Hypothesis

To build a Sequence–to–Sequence model that implements Attention mechanism on the top of Encoder (LSTM) – decoder (LSTM) architecture to be trained to map amazon fine food reviews to its summary.

5. Proposed Seq2Seq Model

We propose a sequence–to–sequence model that applies the Encoder and decoder mechanism. Generally, variants of Recurrent Neural Network (RNN), i.e. Gated Recurrent Units or Long Short Term Memory (LSTM), are preferred as the encoder and decoder components. These variants are capable enough to capture long term dependencies by overcoming the problem of vanishing gradient. We set up the Encoder–Decoder in 2 phases.

i. Training Phase
ii. Inference Phase

The architecture of the model in the inference is different than the architecture used for training. However, the layers that are trained during the training are used in the inference model with a different data-flow.

5.1. Training Phase

In the Training Phase, a vector of a review text is read by an encoder as input. An encoder–LSTM reads one word–vector at each timestep and produces a hidden state (h_i) and a cell state (c_i). In the beginning, the encoder use zeros or random initial states (h_0, c_0).

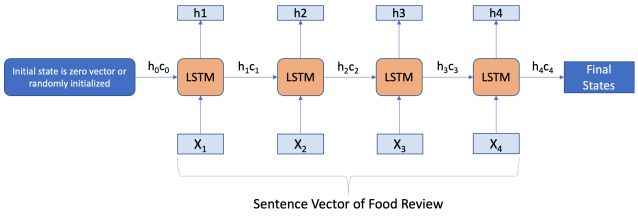


Figure 5.1.1: Encoder Architecture in Training Phase

These hidden values are used as initial states in process of the next time step. This how the LSTM keeps capturing the information that resides in every word at each timestep. The final values of the hidden state(h_n) and the cell state(c_n) will possess the information of the entire word sequence of the review text.

The Decoder uses the final states of the Encoder as initial states (h_0, c_0). The decoder is also an LSTM network that reads the entire target sequence word-by-word and predicts the same sequence offset by one timestep. During the training, the decoder to be trained to predict the next word in the sequence given the previous word. The target sequence is unknown while decoding the test sequence. So, the decoder starts predicting the target sequence by passing the first word into the decoder which would be always the <**start**> token. And the <**end**> token signals the end of the sentence.

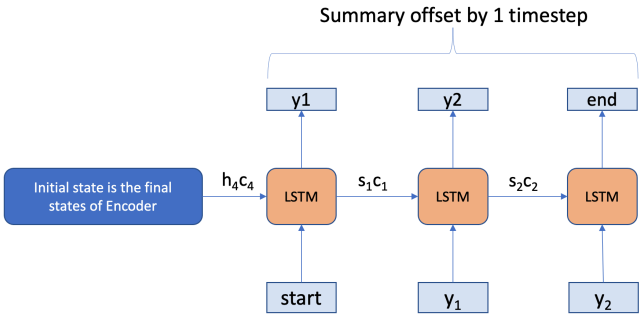


Figure 5.1.2: Decoder Architecture in Training Phase

This simple Encoder-Decoder architecture has some following limitations.

- The encoder converts the entire input sequence into a fixed-length vector and then the decoder predicts the output sequence. This works only for short sequences since the decoder is looking at the entire input sequence for the prediction.
- The problem with long sequences is difficult for the encoder to memorize long sequences into a fixed-length vector [1].

To overcome these limitations, an *Attention Layer* is added on the top of the Encoder and Decoder. The intuition behind the attention mechanism is "How much attention we need to pay to every word in the input sequence for generating a word at timestep t" [1]. By adding an attention layer, instead of looking at all the words in the source sequence, the importance of

specific parts of the source sequence that result in the target sequence can be increased.

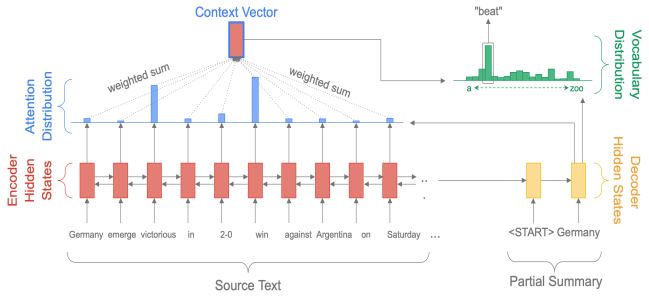


Figure 5.1.3: Attention Layer

5.2. Inference Phase

After training, the model is tested on new source sequences for which the target sequence is unknown. The basic steps to infer the summary of the given unknown review are as follows.

1. Encode the entire input sequence and initialize the decoder with internal states of the encoder
2. Pass <start> token as an input to the decoder
3. Run the decoder for one timestep with the internal states
4. The output will be the probability for the next word. The word with the maximum probability will be selected
5. Pass the sampled word as an input to the decoder in the next timestep and update the internal states with the current time step
6. Repeat steps 3 – 5 until we generate <end> token or hit the maximum length of the target sequence

5.3. Model Architecture

Final model three main components

i. Embedding layer
ii. Stack of LSTM
iii. Attention layer

We have used a stack of more than one LSTM layer to improve the training performance. We also used a pre-trained attention model available on GitHub [4]. More LSTMs helps the model to converge quickly. Figure 5.3 shows a detailed architecture with all hyperparameters.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 80)]	0	
embedding (Embedding)	(None, 80, 512)	2619904	input_1[0][0]
lstm (LSTM)	[(None, 80, 512), (N	2099200	embedding[0][0]
input_2 (InputLayer)	[(None, None)]	0	
lstm_1 (LSTM)	[(None, 80, 512), (N	2099200	lstm[0][0]
embedding_1 (Embedding)	(None, None, 512)	550400	input_2[0][0]
lstm_2 (LSTM)	[(None, 80, 512), (N	2099200	lstm_1[0][0]
lstm_3 (LSTM)	[(None, None, 512),	2099200	embedding_1[0][0] lstm_2[0][1] lstm_2[0][2]
attention_layer (AttentionLayer	((None, None, 512),	524800	lstm_2[0][0] lstm_3[0][0]
concat_layer (Concatenate)	(None, None, 1024)	0	lstm_3[0][0] attention_layer[0][0]
time_distributed (TimeDistribut	(None, None, 1075)	1101875	concat_layer[0][0]

Total params: 13,193,779
Trainable params: 13,193,779
Non-trainable params: 0

Figure 5.3: Final Model Architecture

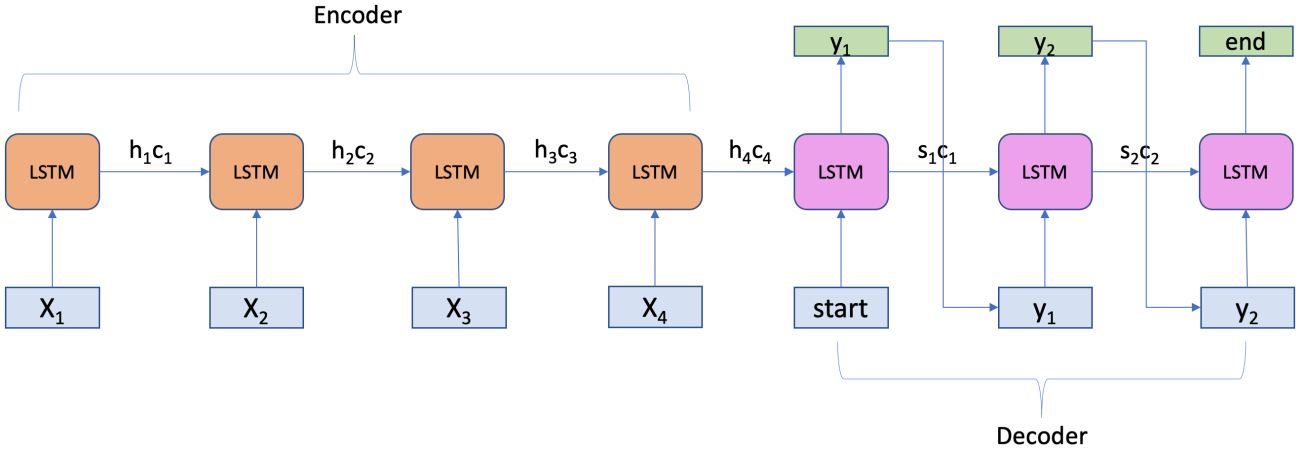


Figure 5.2: Architecture of Inference Phase

6. Model Evaluation

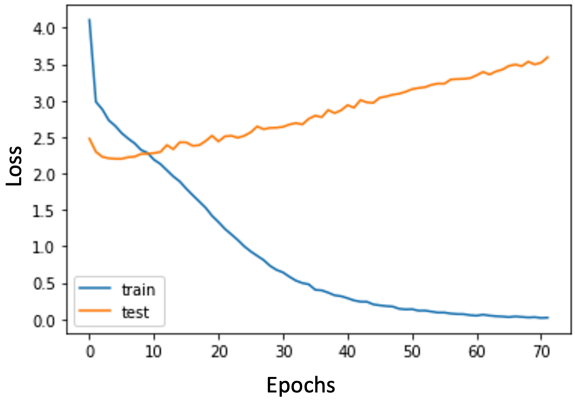
To evaluate our model's performance on testing data, we used the BLEU score technique. **BLEU (bilingual evaluation understudy)** is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another [5]. Quality is considered to be the correspondence between a machine's output and that of a human: "the closer a machine translation is to a professional human translation, the better it is" – this is the central idea behind BLEU [4]. We used the principle of BLEU score to compare the model generated summary with the original summary. We used a BLEU score function available at an opensource nltk python library[5].

7. Results and Discussion

We derived a sequence of training and testing outcomes by applying different hyperparameters and various lengths of the dataset. As per figure 6.1, we use a different approach to train and test out the model. We tried to train our model on the data set of 100,000 reviews. However, because of the limited computational power and available devices, we could not train our model on 100k reviews in a limited time. To evaluate our approach, we trained and evaluate our model in one small dataset of only 1000 reviews. This lets us discover the most optimized and ideal sequence to sequence architecture.

As shown in table 7.1, the embedding dimension has the highest impact on the training results and training time. The more higher Embedding dimensions are the more time-consuming model-training will be. However, less embedding dimensions do not help to train on the food reviews. We have experienced that; 256 or more embedding dimension are required to train model efficiently.

We have also realized that to train our model on higher amount of data will help the model to perform well on unseen reviews. But we due to limited time and device efficiency, we are able to train our model only on 1000 with considerable training results. On the other hand, training the model only on 1000 food reviews results in overfitting on training data. The following graph shows significant decrement in training loss with negligible.



Size of Data	Embedding Dim	No. LSTM used in the Stack of LSTM	Epochs	Batch size	Average Loss Value	Average Training Time for 1 epoch	Expected time to train the model
100, 000	512	3	25	512	1.2798	46 mins	20 hours
100, 000	64	1	25	256	3.0033	24 mins	10 hours
100, 000	128	2	50	512	2.245	31 mins	26 hours
1000	512	3	150	64	0.0321	74 secs	3 hours
1000	1024	3	50	64	0.01945	16 mins	14 hours
1000	512	3	72	64	0.028	74 secs	88 mins

Table 7.1

Approach	Training data size	Testing data size	Training Loss	Testing Loss	Training Bleu Score (unigram)	Testing Bleu Score (unigram)
1st	80,000	20,000	2.3424	3.021	0.01098	0.0065
2nd	800	200	0.024	1.190	0.6789	0.3480

Table 7.2

```
Review: fresh great way get little chocolate life without million calories taste like
chocolate pudding
Original Summary: omg best chocolate jelly belly
Predicted Summary:  omg best chocolate jelly belly <unk> <unk> <unk> <unk>


Review: coffee finely processed flavor alright nescafe mount hagen better bought try
something two probably get slight vitaminy taste overwhelming also bit pricey get
Original Summary: it ok
Predicted Summary:  it ok <unk> <unk> <unk> <unk> <unk> <unk> <unk>


Review: ordered dad fathers day ate whole box within two days said tasted really good
want order haha
Original Summary: dad really liked these
Predicted Summary:  dad really liked these <unk> <unk> <unk> <unk> <unk>


Review: save little money buying larger quantities opposed smaller bottles like taste
agave cross honey maple good pies cereal frozen yogurt etc
Original Summary: agave nectar
Predicted Summary:  agave nectar <unk> <unk> <unk> <unk> <unk> <unk> <unk>


Review: munchkin fresh food feeder okay product kind like sucking food cloth daughter
interested little eventually lost interest may introduce back see reacts
Original Summary: was not that impressed
Predicted Summary:  was not that impressed <unk> <unk> <unk> <unk> <unk>
```

Figure 7.3: Results

8. Applications

These are some use cases where automatic summarization can be used across the enterprise.

1) Media Monitoring
2) Newsletters
3) Search marketing
4) Search Engine Optimization (SEO)
5) Financial Research
6) Legal Contract analysis
7) Question answering and bots
8) Video scripting
9) Books and Literature
10) E-learning and Class assignments
11) Help desk and customer support
12) Programming languages
13) Automated content creation

9. Conclusion

The proposed sequence-to-sequence model is efficient to learn text summarization for amazon fine food reviews. It can be further improved by training it on the high volume of data with high computational devices.

10. References

[1] Comprehensive Guide to Text Summarization using Deep Learning in Python – Arvind Pai https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/

[2] Allahyari, M., Pouriyeh, S., Assefi, M., Safaei, S., Trippe, E. D., Gutierrez, J. B., & Kochut, K. (2017). Text summarization techniques: a brief survey. *arXiv preprint arXiv:1707.02268*.

[3] Deep Reinforcement Learning for Sequence-to-Sequence Models - Yaser Keneshloo, Tian Shi, Naren Ramakrishnan, Chandan K. Reddy

[4] https://github.com/thushv89/attention_keras/blob/master/layers/attention.py

[5] https://en.wikipedia.org/wiki/BLEU