Enhancing abstractive text summarization using an extractive layer

Amit Dharmadhikari
amit.dharmadhikari
@stonybrook.edu

Sakshi Gupta
sakshi.gupta
@stonybrook.edu

Shoaib Sheriff
shoaib.sheriff
@stonybrook.edu

Abstract

This paper puts forth a methodology to produce a summary from long sequences of unstructured text. The proposed method attempts to enhance abstractive summarization methods by augmenting it with an extractive summarization layer first. We compare the performance of the proposed model against the baseline model - a simple abstractive model. The datatset used in CNN/Dailymail datatset.

1 Introduction

In today's world, unstructured content accounts for around 95 percent of the digital information available online. With this abundance of large text documents, it is a necessity to have a mechanism that helps us extract important information from the text efficiently and effectively. Text Summarization is the process of extracting meaningful information which is shorter than the given text while still preserving its meaning. There can be two types of summarization - Single-document summarization and Multi-document summarization. Both suffer from problems like retaining the quality of the original text, dealing with long sentences that do not have relevant information and evaluating the end results of such processes against the ground truth.

There are basically two major types of approaches to perform text summarization: extractive summarization and abstractive summarization. The extractive approach basically selects a subset of sentences from the original text. It ranks sentences by importance and chooses a subset of them to form a summary[1].The abstractive approach on the other hand tries to generate new sentences for the summary instead of picking them up directly from the original text[2].

Extractive summarization, however, has many shortcomings. Since it picks up sentences directly from the original text, it has no flexibility and cannot introduce paraphrases. Sentences also may not flow in to each other. This problem is solved by abstractive summarization. But, it needs a complex system that can take as input large documents and generate new and relevant words that results in a well defined summary of the original text.

As mentioned above, both the approaches suffer from some problems and hence many combination approaches have been tried out. One of them is the Pointer-Generator Approach. The Pointer-Generator system basically performs *pointing* to get words directly from the original text and *generating* which generates new words and paraphrases to provide coherence in the summary. This system also has a *coverage* factor that keeps track of repetitions, a major problem that abstractive summarization suffers from[4]. Another simple approach is the Extract-Then-Abstract approach which is basically a pipeline system where we provide as input the original text to the extractive model and then give that output as an input to the abstractive model in order to get a better representation[5].

There are two main classes of evaluation metrics for text summarization - intrinsic and extrinsic.[3] Intrinsic measures evaluate how good a summary is by itself. Extrinsic measures use external tasks like question answering and information retrieval to evaluate how good a summary is. We are concentrating on intrinsic measures to evaluate our results. Intrinsic measures mainly include text quality evaluation and content evaluation. Text quality evaluation is concerned with maintaining correct grammar and avoiding redundancy. Content evaluation measures how good the summary is at capturing the essence of the text.

2 Our Task

2.1 Baseline Model

For the baseline model, we use a Pointer Generator network. It provides a mechanism of combining extractive and abstractive summarization. Traditionally, neural sequence-to-sequence summarization networks use an encoder - decoder architecture for summary generation. The encoder uses a bi-directional LSTM to produce the current representation, which the decoder then uses to generate the attention distribution over the source text[8].
Instead of generating each word, however, a Pointer Generator Network lets the model choose between generating a word or just extracting a word. The vocabulary distribution generates the word we may want to generate. The attention distribution indicates the word we want to point to. Using the switching probability, we combine the above two probabilities to a final distribution.

More precisely, the probability to generate a word, w at timestep t is given by

$$P(w) = p_g * P_{vocab}(w) + (1 - p_g) * \sum_{i:wi=w} a_i^t \quad (1)$$

where, p_g is the probability to generate a new word from the vocabulary, p_{vocab} is the probability distribution over the vocabulary and a_i is the attention weight for the ith word token.
To avoid repetition, coverage is used. Any part of the original text that is already covered has high coverage. The loss function is designed to ignore texts with large coverage.

2.2 The issues

The issues that we could identify with the baseline model were:

- The system fails to pick out the important content when there are a lot of ideas present in the main text.
- Despite having a complex architecture, the system fails to process long sentences/sequences.
- When blog/opinionated text pieces are fed to the system, it just gives out random sentences from the story as the summary.
- Trips over sentences with quotes in the original text and mistakes it to be the important sentences that need to be picked for summarization.

3 Our Approach

We concentrated on two main approaches. In the first approach, we built a pipeline system wherein we perform extraction on the input document and then abstraction on the previously generated output to get a final summary. In second approach, we combine the two steps in a more subtle way.

3.1 Idea 1: Extract to Abstract Model

For the extractive part, we use text-rank algorithm. It uses the PageRank algorithm for text summarization.[6] PageRank is the algorithm which was used by Google for ranking the relevance of the results of search queries. The basic idea is that the web pages which have a high number of links pointing to them are more important than others. In order to apply this algorithm to text summarization, we map the web pages to sentences and the number of links to the cosine similarities between sentences. Sentences which have a high number of sentences similar to them will capture the gist of the input paragraph, and hence, they must be included in the summary.

The extractive model gives an importance score for all the sentences. We then omit k least important sentences from the summary, while ensuring original sentence order is maintained. The output of this is passed to abstractive summarization.

For the abstractive model, we use the pointer-generator network. The output of the abstractive model is the final summary.

3.2 Idea 2: Entity pointers

In this case, we use a more subtler mechanism to combine the extractive and abstractive models. Instead of breaking the flow of information across sentences like above, we try to learn entity level information.
Since we use a news story dataset, it may be assumed that a majority of sentences would be factual. If a sentence is ranked important, it is probably because it talks about important entities. This is the intuition which this model works on. As above, we use the extractive summarization to rank important sentences. Next, for the important

sentences, we use a Named Entity Recognizer to identify constituent entities. The importance of a sentence is divided equally among its entities. Note that, a single entity can occur in more than one sentence. So, for each original story text, we have a mapping of entity to a number indicating its importance. Normalization ensures that importance is unaffected by the length of the original text document.

We then use the entity importance data inside the pointer generator. When we generate a probability of a word given by eq:1, and the word is an entity, we may use this importance information. This increases the chances of this word appearing in the summary and achieves a better summary, than the baseline model. eq:1 can be updated as

$$\begin{aligned} &\text{P(w)} = \text{p}_g * P_{vocab}(w) + \\ &(1 - \text{p}_g) * \sum_{i:wi=w} a_i^t * (1 + w_m * m_i) \end{aligned} \quad (2)$$

where, m_i is the importance value, if ith token is an entity, otheriwse 0
w_m is a learnable parameter.

3.3 Implementation Details

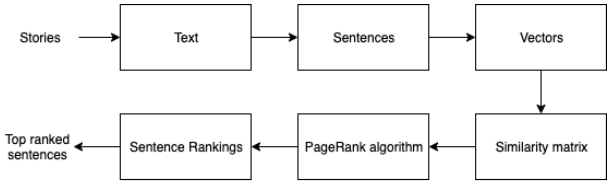

Figure 1: Extractive Model

3.3.1 Idea 1: Extract to Abstract Model

The extractive model is implemented in python. For every story, we first separate the main text from the annotated summary. Every sentence is cleaned by removing stop words and then lemmatizing the words. Next, we aim to build a similarity matrix. First, We make a list of all unique words across two sentences. Each sentence is then converted in to 1-hot encoded form, wherein every index represents the count of the word, representing that index, in the sentence. A cosine similarity between the two words is used to calculate strength of connection. Next, we use the page rank algorithm, repeatedly until convergence, to give a vector that represents the importance of sentences. We remove 40% least important sentences from the original text and pass it to the pointer generator network.

The pointer generator network uses an encoder-decoder architecture internally. The encoder is a Bi-LSTM model and decoder is a single forward LSTM. at every time step t, we pass in a word from the original story text. This outputs an encoded hidden state. The maximum sequence length can be set as an argument in the python program. Similarly, we also generate a decoder output at every time step. During training, the decoder input is the previous word of the reference summary; at test time it is the previous word emitted by the decoder.
We use the encoder and decoder state at every time step to generate the attention at every time step. We also use the coverage vector in the above, since it helps to avoid word repetitions in summary generation. Once we have the attention weights, we use it to calculate the context vector.
We then use the context vector and the decoder state to generate two probabilities - probability of generation and probability over the vocabulary. Once we have these 2 terms, we can use eq:1 to generate the probability for words across the extended vocabulary and output the most probable word.

We use pretrained weights for the pointer generator (trained on whole stories, available to download online) as starting points and train the network with our training data for 15 epochs. Note that, the dataset is divided in a 70/10/20 fashion in to training, validation and test data in this case.

3.3.2 Idea 2: Entity Pointers

The same extractive model is used as from above. But afterwards, we pick the top 60 percent sentences and run them through a Named Entity Recognizer. We experimented with Stanford-nlp, but finally decided to pick Spacy for this task. We divide the sentence scores equally among the entities and once all sentences are processed, normalize across entities.

The pointer-generator works pretty much as above, but also takes in the information about entity importances. We use this information in eq:2 to generate a more suitable probable distribution and a better summary as a result.
Same training setup is used as above. But in here, we pass whole stories as input (instead of shortened). we train the model for around 15 epochs so that it has an opportunity to learn the new parameters.

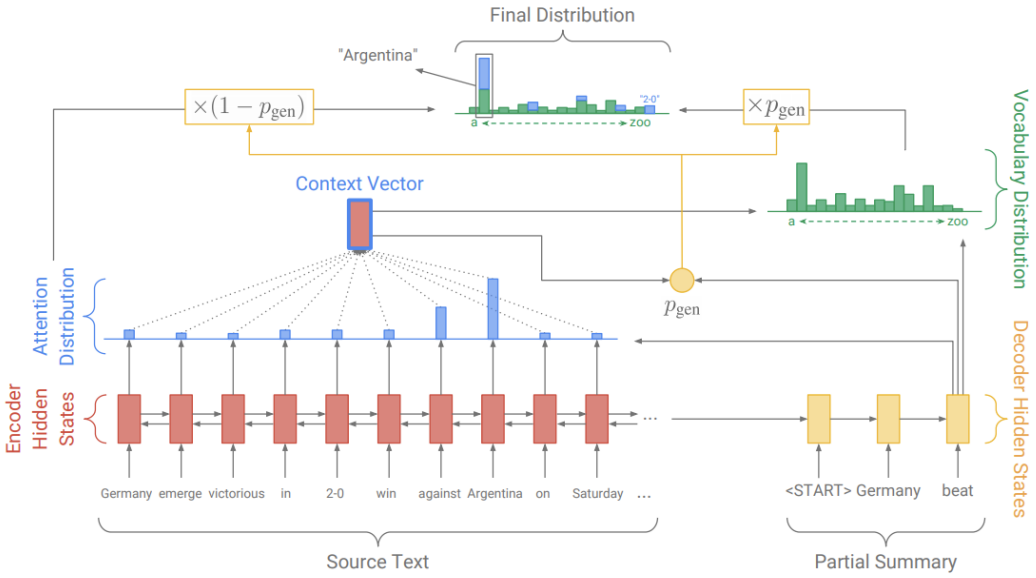

Figure 2: Abstractive Model

4 Evaluation

4.1 Dataset Details

The dataset used is the CNN Dataset. It was created by Hermann(Google Brain) in 2015. The dataset consists of 10000 long CNN news articles(an average of 800 words) along with annotated summaries[9]. It originally has around 90K documents and 380K questions. Since it would be pointless to run extractive summarization on small stories, we drop stories with less than 20 sentences. Hence, our real dataset size has 7054 stories.

4.2 Evaluation Measures

4.2.1 BLEU

BLEU (bilingual evaluation understudy) is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another. 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. BLEU was one of the first metrics to claim a high correlation with human judgements of quality, and remains one of the most popular automated and inexpensive metrics.

Since summarization also uses encoder-decoder model like translation, we can use BLEU for evaluation. Scores are calculated by comparing generated summary sentences with annotated summary. Those scores are then averaged over the whole corpus to reach an estimate of summarization quality. Intelligibility or grammatical correctness are not taken into account.

BLEU's output is always a number between 0 and 1. This value indicates how similar the candidate text is to the reference texts, with values closer to 1 representing more similar texts.

4.2.2 ROUGE

ROUGE, or Recall-Oriented Understudy for Gisting Evaluation, is a set of metrics and a software package used for evaluating automatic summarization and machine translation software in natural language processing. The metrics compare an automatically produced summary or translation against a reference or a set of references (human-produced) summary or translation.

The following five evaluation metrics are available:

- ROUGE-N: Overlap of N-grams between the system and reference summaries.
  - ROUGE-1 refers to the overlap of unigram (each word) between the system and reference summaries.

  - ROUGE-2 refers to the overlap of bigrams between the system and reference summaries.

- ROUGE-L: Longest Common Subsequence (LCS) based statistics. Longest common subsequence problem takes into account sentence level structure similarity naturally and identifies longest co-occurring in sequence n-grams automatically.
- ROUGE-W: Weighted LCS-based statistics that favors consecutive LCSes .
- ROUGE-S: Skip-bigram based co-occurrence statistics. Skip-bigram is any pair of words in their sentence order.
- ROUGE-SU: Skip-bigram plus unigram-based co-occurrence statistics.

4.3 Results

Even though, we use 2 approaches as described above, the results from the second setup out perform the first. Hence, all following results are obtained with the 'NER links' setup.

We used two metrics to measure our results. First is BLEU (bilingual evaluation understudy) and the second is ROUGE (Recall-Oriented Understudy for Gisting Evaluation).

4.3.1 Result 1

Reference summary:
rostenkowski first entered congress in 1959 . he became chairman of the tax-writing ways and means committee in 1981 . rostenkowsi was defeated in the republican landslide of 1994 .

Baseline model:
former illinois congressman dan rostenkowski died in wisconsin after an extended illness . he died in wisconsin after an extended illness . he helped pass a controversial expansion of medicare designed to protect seniors against catastrophic medical expenses .

Our model:
rostenkowski was defeated in the republican landslide of 1994 . he died in wisconsin after an extended illness , mell 's office said . rostenkowski was chairman of the tax-writing ways and means committee in 1981 .

4.3.2 Result 2

Reference summary:
calls for lee wan-koo to resign began after south korean tycoon sung woan-jong was found hanging from a tree in seoul . sung , who was under investigation for fraud and bribery , left a note listing names and amounts of cash given to top officials .

Baseline model:
south korea 's prime minister lee wan-koo offered to resign on monday . he has transferred his role of chairing cabinet meetings to the deputy prime minister . calls for lee to resign began after south korean tycoon sung woan-jong

Our model:
calls for lee to resign began after south korean tycoon sung woan-jong was found hanging from a tree . sung , who was under investigation for fraud and bribery , left a note listing names and amounts of cash given

4.3.3 Result 3

Reference summary:
how " the girl from scotland " won her first olympic gold medal . why sydney is far more than just a place to race : from rugged headlands to bustling harbors . robertson recalls battling back from 16th place to secure the win .

Baseline model:
" so what does it feel like ? " " hmmm ... cool i suppose , " i did n't know what to say , how could i not feel " it ? " was n't it the greatest memory

Our model:
the favorite for the prestigious rolex sydney to hobart race . the next day we were out on the harbor filming for mainsail , onboard the mighty comanche , the favorite for the prestigious rolex sydney to hobart race .

4.3.4 Final Result Statistics

We have used BLEU, ROUGE-1, ROUGE-2 and ROUGE-L to evaluate our output summaries. The results are as follows:

- BLEU: Our model produced better summaries than the baseline model for 2463/7054

articles, or rather, 35% of the articles.

- ROUGE-1: Our model produced better summaries than the baseline model for 2396/7054 articles i.e. 34% of the articles.
- ROUGE-2: Our model produced better summaries than the baseline model for 2053/7054 articles i.e. 29% of the articles.
- ROUGE-L: Our model produced better summaries than the baseline model for 2448/7054 articles, or rather, 35% of the articles.

4.4 Analysis

We found out that our model performs significantly better than the baseline model when the article has a lot of quotes. The baseline model focuses too much on the unimportant quotes and often just copies the quotes verbatim. Our model takes the right quotes and summarizes the content effectively.

Further, we found out that the baseline model sometimes puts too much focus on opinion rather than fact. Sometimes, when there is a reporter mentioning his/her opinion, which is not the most important thing in the article, the baseline model puts too much focus on what the reporter is saying. Our model effectively finds the gist of the article and doesn't put undue importance on opinion.

Section 4.3.1 is an example of how our model produces a better summary than the baseline model:

We see that the baseline model focuses too much on the death of Congressman Rostenkowski, which is not even mentioned in the reference summary. The fact that he was a chairman of a committee is captured in our model, whereas the baseline model fails at this.

Section 4.3.2 is another such example. We see that the summary produced by our model closely resembles the one produced by the baseline model, whereas the baseline model doesn't focus on the South Korean tycoon found hanging. Moreover, the baseline odel doesn't even produce a grammatically and semantically correct output.

Section 4.3.3 is an example which showcases the problem with articles containing a lot of quotes. The baseline model includes just quotes, and that too the unimportant quotes, whereas our model is able to produce a better and more coherent output, and it is able to capture the topics like "race", "sailing" and "Sydney" which the baseline model isn't.

All in all, the performance of the model is closely tied to the performance of the extractive summarization. The proposed model fails to improve or even underperforms when the extractive part fails to rank sentences well.

4.5 Code

Our code mainly consists of 4 modules - preprocessing/filtering, extractive summarization, abstractive summarization and results section. The code can be found here. Please check the README for more details. We have also include links to all results at the end.

5 Conclusion

This project aimed to tackle the problem of text summarization. While a lot of progress has been made in abstractive summarization, this project aims to fix some of the issues we discovered. Extractive summarization has some benefits and we attempt to use these to aid the abstractive process.

Proposed model manages to produce better summaries for around 35% of the stories. For future work, the quality of the extractive network used can be bettered. Better models like PriorSum[10] and SummaRuNNer[11] can be used.

6 References

[1] Ramesh Nallapati, Feifei Zhai , Bowen Zhou. *SummaRuNNer: A Recurrent Neural Network based Sequence Model for Extractive Summarization of Documents.*

[2]Yen-Chun Chen and Mohit Bansal. *Fast Abstractive Summarization with Reinforce-Selected Sentence Rewriting.*

[3] Josef Steinberger, Karel Jezek. Evaluation Measures for Text Summarization.

[4]Abigail See, Peter J. Liu and Christopher D. Manning. *Get To The Point: Summarization with Pointer-Generator Networks.*

[5]Peter J. Liu,Mohammad Saleh,Etienne Pot,Ben Goodrich, Ryan Sepassi, Łukasz Kaiser, Noam Shazeer. *Generating wikipedia by summarizing long sequences.*

[6]https://github.com/amanraj209/text-summarization

[7]Leo Laugier,Evan Thompson and Alexandros Vlissidis. *Extractive Document Summarization Using Convolutional Neural Networks - Reimplementation*

[8]Qian Chen,Xiaodan Zhu, Zhenhua Ling,Si Wei,Hui Jiang. *Distraction-Based Neural Networks for Document Summarization*

[9]https://cs.nyu.edu/ kcho/DMQA/

[10] Ziqiang Cao, Furu Wei, Sujian Li, Wenjie Li, Ming Zhou, Houfeng Wang. *Learning Summary Prior Representation for Extractive Summarization*

[11] Ramesh Nallapati, Feifei Zhai, Bowen Zhou. *SummaRuNNer: A Recurrent Neural Network based Sequence Model for Extractive Summarization of Documents*