CS230

Abstractive Summarization on COVID-19 Publications with BART

Jiachen Xu
jx2318@stanford.edu

1 Introduction

COVID-19 pandemic triggered a proliferation of scientific literature to provide valuable insights into the disease. However, the sheer volume poses a challenge for researchers and medical professionals to quickly extract knowledge of their interest. To mitigate this issue, this project aims to applying a deep-learning-based pipeline to 1. cluster all research papers based on themes 2. generate the abstractive text summarization for each group of scientific publications.

2 Literature Review

Deep learning models have been applied on multiple natural language processing tasks, like sentiment analysis, name entity tagging, machine translation etc.. Among all these tasks, summarization is one of popular topic. Generally speaking, there have two approaches to generate summarization, either extractive methods or abstractive methods. Extractive summarization aims to extract a brunch of core sentences from the whole paragraph and concatenate them to generate a single summary. The advantages of this methods are easier to implementation, more flexible strategy (NLP based or deep learning based) and helpful to highlight the important sentences from a long paragraph. On the contrary, sentences extracted from the original texts are concatenated directly to generate the summary, which makes it harder to understand without adequate context. Zhong, etc.,[1] proposed a model applying neural extractive summarization systems to match the source and target text semantically, and achieved great accuracy. Some other advanced models with encoder-decoder [2], auto-regressor encoder [3], reinforcement learning [4] architectures are achieved pretty nice performance as well.

On the contrary, abstractive summarization is more challenge, which aims to generate a new paragraph as the summary based on the original text. Some deep Learning architectures, like encoder-decoder, transformer, etc., have invisible advantages on solving this issue. Especially bidirectional encoder could encode the whole paragraph and learn its meaning together, and then the decoder step could take this output form encoder block to generate the summary. For instance, BertSum [5] based on the Bidirectional Encoder Representations from Transformers (BERT) architecture introduced a novel document-level encoder to express the semantics of a document and obtain representations for its sentences. When fine tuning the model, the encoder - decoder architecture could works on alleviating the mismatch generated summary and target achieving better abstractive summarization. Except these models, PNBERT [6], prophetnet [7], etcs. Also got pretty nice performance on this task.

CS230: Deep Learning, Spring 2020, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

3 Summarization

Since the pipeline for topic clustering is relative normal. This part will focus on introducing different experiments on summarization.

3.1 Baseline Model: text rank

The baseline model is a extractive summarization algorithm, named text rank [8]. This is a graph-based ranking algorithms inspired by page rank algorithm. The algorithm tries to decide the importance of a vertex, either a sentence or a word, within a graph based on global information recursively draw from the entire graph. The higher scores of a vertex, more important it is. The score of vertex i V_i is defined as

$$S(V_i) = (1-d) + d * \sum_{j \in In(V_i)} \frac{1}{|Out(V_j)|} S(V_j)$$

where $In(V_i)$ mentioned the set of vertices that point to V_i and $Out(V_i)$ is the set of vertices that vertex V_i points to. d is a damping factor that between 0 and 1 to integrate into the model the probability of jumping from a given vertex to another random vertex in the graph. For a given text, the edge could defined like co-occurrence relation, etc..

3.2 BART Model

The deep learning model I implemented and improved for this project is BART model [9]. This is a denoising auto-encoder seq2seq model pre-training for natural language generation, translation and comprehension. This model could be treated as a general BERT model because it users a standard seq2seq/ machine translation architecture with a bidirectional encoder (like BERT) and a left-to-right decoder (like GPT). The training strategy of this model is firstly corrupting text with an arbitrary noising function and then learning a model to reconstruct the original text. The noising approached includes, random shuffling the order of the original sentences and using a novel in-filling scheme, etc..**??**

3.3 Pointer-Generator

Pointer-Generator [10] is another quite popular technique applied on abstractive text summarization. This architecture generates a pointer probability from the context vector, decoder state and input to decoder. This probability is used to control the probability of generating words from the vocabulary, versus copying words from the source text. It is a kind of common sense that the words in summary are highly possible from the original input paragraph. The final vocabulary distribution output is the weighted sum of attention distribution and vocabulary distribution. The formula for generating the generation probability is defined as

$$p_{gen} = \sigma(w_h^T * h_t^* + w_s^T * s_t + w_x^T * x_t + b_{ptr})$$

where h_t^* represents the context vector, s_t represents the decoder state and x_t represents the decoder input. The final output of the vocabulary distribution is calculated as

$$P(w) = p_{gen} P_{vocab}(w) + (1 - p_{gen}) \sum_{w_i = w} a_i^t$$

where a^t is the attention distribution of the encoder with respect to the decoder.

3.4 BART + Pointer-Generator

This is the innovation techniques I generated for this project. I added a pointer-generator layer at the top of the BART model with the objective to combining the advantage of these two techniques. One methods to combine these two architecture is shown in the plot. 2 [11]

The methods I applied is firstly used decoder output represent the decoder state from the previous pointer-generator modules as s_t . I calculated the attention between the decoder output s_t and encoder output to get the attention distribution a_t. This feature try to measure how strongly the decoder output correlated to the encoder output h_t. Rather than the multi-head attention, I applied simple

attention modules for quick training. And then I generated the point probability based on the attention distribution a_t and decoder input x_t. After that, just apply the point probability on the decoder output and attention distribution. The formula is given below:

$$a_t = Attention(h_t, s_t)$$
$$p_{gen} = \sigma(w_h h_t + w_s s_t + w_x x_t + b_{gen})$$
$$P(w) = p_{gen} P_{vocab}(w) + (1 - p_{gen}) a_t$$

4 Dataset and Preprocessing

The dataset "COVID-19 Open Research Dataset Challenge (CORD-19)" from Kaggle [12] was used. It contains 59,887 scholarly articles, including over 41,000 with full text, about COVID-19, SARS-CoV-2, and related coronaviruses. Each paper has been split into different chunks including authors, abstract, body text and reference, saved as a JSON file under different keys.

However, the data set is not as clean as we expect. Some of the articles have missing values in title, abstract, authors and affiliations. For instance, for the 2,670 papers from biorxiv, around 4% articles do not have a title, and 12% articles do not have an abstract. Since the following analysis is mainly based on articles' titles and abstracts, we dropped those articles at the preprocessing stage. Also, we noticed that not all papers are in English. We detected some data in Italian, and removed them from the overall training data as well.

We use the word cloud in Figure 3 to have a very generic idea about the overall word distribution of the data. The word cloud is generated from the abstract of the papers. Based on it, we could infer the potential topics of the papers, like biology research on protein, patient case studies, infection analysis, data modeling, etc.

5 Experiments

5.1 Clustering

The clustering pipeline is first vectorize the text and then grouping with the k-means methods. Three experiments on clustering were done, grouping based on title only, abstract only or title and abstract. We noticed that phrases were also helpful for clustering the articles, hence set ngram = $(1, 3)$. Since the number of clusters are quite tricky to decide, we experiments cluster number from 2 to 30, and utilized elbow plot to determine the number of clusters.

Based on the experiments, the result of clustering only based on title is not reasonable, because titles of the papers only contain little information. The results of clustering based on abstract only and on both titles and abstracts seem quite similar, which may because the information contained in titles are always covered in the abstract. At the end of the day, we decided to use clusters from abstracts only and selected 9 clusters. The elbow plot and distribution of the clusters are shown Figure 4.

5.2 Baseline Summarization

After clustering with k-means, text rank has been applied to first extract key words from each cluster in order to evaluate the cluster result and then extract the key sentences from each clustering. Top 3 keywords from 4 selected groups are given in Table 1.

Group 1	Group 2	Group 3	Group 4
covid-19 patients	sars cov-2	host cell proteins	daily reported case numbers
adult covid-19 patients	sars infections	rna virus infection	epidemic models
non-hospitalized patients	human sars	rna virus	transmission models

Table 1: Key Words from 4 Clusters

Based on the table below, we thought that this method has a pretty good performance on extracting keywords. However, the extracted core sentence is not quite satisfying. This is one example (take one paper's abstract as an example.) The highlight sentence is the sentence extracted as the summary for the whole paragraph.

> Abstract: The asymptomatic carriers identified from close contacts were prone to be mildly ill during hospitalization. However, the communicable period could be up to three weeks and the communicated patients could develop severe illness.**These results highlighted the importance of close contact tracing and longitudinally surveillance via virus nucleic acid tests.** Further isolation recommendation and continuous nucleic acid tests may also be recommended to the patients discharged.

There are two main drawbacks based on our observation.

- The summarization methods is extractive. For a single article, this method may have good performance, but since we want to summarize a group of articles, the results from extractive summarization methods may be hard to understood, because it extracts sentence from different articles.
- The results are only one or a few sentence(s) from the whole paragraph, which is hard to understand without any context.

5.3 BART Model

5.3.1 Pretained BART Model

We applied the open source code from huggingface [13] to implement the pre-trained BART model on generating the abstractive summary. The generated summary for the previous example is given below:

> Summarize: **The asymptomatic carriers identified from close contacts were prone to be mildly ill during hospitalization. The communicable period could be up to three weeks and the communicated patients could develop severe illness. These results highlighted the importance of close contact tracing and longitudinally surveillance via virus nucleic acid tests.**

Based on the results, we have several foundings:

- The summary results generated from pretrained BART model cut off some unnecessary part from the original articles (e.x. However, etc..) but still readable.
- May because this model is pretrained with the CNN news data. When the input articles includes some biology noun, which is usually the core topics for the whole paragraph and want to include as a part of the summary, the model tends to miss out this parts.
- most of the generated summaries, (not like the shown ones), is quite short. Usually one ortwo sentences.

In order to address these challenges, we thought one strategy is to fine-tune the BART model.

5.3.2 Finetune BART Model

I modified the finetune.py code from transformers to finetune the BART model with the COVID paper data. We selected the abstract as the training data, title as the target data. The results did not makes sense at all and the model tends to generate same summary for all inputs.

> Summarize: **'VID- of in and19 the CO: S2CoVARS for aav withirus to on, China A'**

The summary seems like the combination of some high-frequent words. We thought this may because

- the size of the finetune data was quite small, only 1667 articles.
- because do not have the ground truth for this data set, I applied the title as the target output and abstract as the input source. And title usually a quite high level summary of the whole articles to include the term and pretty short.

5.3.3 BART + pointer-generator Model

The pointer-generator layer is added after generating the output from decoder. I freeze all parameters' weights in encoder and decoder block and only fine-tuned the added pointer-generator layers. I applied the CNN-Daily news data as the training data for two reasons: 1. the COVID research paper data is too limited. 2. the original weights of the model is trained with the CNN-Daily news data. And if applied the original model directly on generating summary for the COVID research paper data, the results were not bad. Hence, it is possible to train the model weights with the CNN Daily data and then applied it on another data set. There have several BART configuration selections. I applied the BART-Large-CNN at the beginning but met the CUDA memory error. In other to avoid the memory issue, I applied the 'sshleifer/bart-tiny-random' at the end. This is a light version of the original model. Some main difference include, the embedding size is 24 , rather than 1024. And have 2 encoder, decoder block separately, rather than 11 block in BART-LARGE-CNN model.
The training time with 1 GPU p2.xlarge instance is around 5 hours for one epoch. The batch size set as 1. (stochastic gradient descent)

> Summarize: **0,"b' to be a the his her of in on. He says was is has been not have are with by for at from up out as it, and"': NEW will they he had were that she said say after over him into their time-year agoś'"**

	F-score	Precision	Recall
ROUGE 1	0.1835	0.2055	0.1760
ROUGE 2	0.0086	0.0099	0.0081
ROUGE l	0.1636	0.1687	0.1670

Table 2: Rouge Scores

The ROUGE score on the CNN-Daily data set is shown in the tale. The score is pretty low. I thought this may because I only fine tuned the model one epoch / the implementation has some mistakes. Also, I applied the simple Attention block, and multi-head attention may have better performance.

6 Conclusion and Next Step

From this project, I realized that applying deep learning model on summarising the text could get pretty nice performance, even use the pre-trained model.
However, I realized some drawbacks of the current pipeline and could be improved in the next step. One largest issue is that the current BART model, and most of the deep learning based model has limitation on the input sequence length. For instance, for the BERT based model, the maximum input source length is 1024. However, in the real life, when input paragraphs /articles are longer than 1024 characters, the summarization task is more useful. There have two methods to solve this issue.

- use the extractive methods to extract the core sentence firstly and then applied the abstractive summarization. However, generating the abstractive summary based extracted sentences without context could not guarantee a good performance considering the concatenate core sentence itself may not make sense.
- applied the LSTM modules sequential on the input source, (BERT-AL) This algorithm is just new and still open-reviewed without open source code. Hence, it is pretty hard to evaluate its robustness on different data sets correctly.

Also, because of the time and source limitation, I only fine-tuned the BART pointer-generator model 1 epoch and applied multi-head attention. In the next step, I planned to train the model with more epochs and replace the attention block to multi-head attention when calculating the attention distribution between the encoder output and decider output to see whether its performance could have further improvements.

References

[1] Ming Zhong, Pengfei Liu, Yiran Chen, Danqing Wang, Xipeng Qiu, and Xuanjing Huang. Extractive summarization as text matching. *arXiv preprint arXiv:2004.08795*, 2020.

[2] Qingyu Zhou, Nan Yang, Furu Wei, Shaohan Huang, Ming Zhou, and Tiejun Zhao. Neural document summarization by jointly learning to score and select sentences. *arXiv preprint arXiv:1807.02305*, 2018.

[3] Chris Kedzie, Kathleen McKeown, and Hal Daume III. Content selection in deep learning models of summarization. *arXiv preprint arXiv:1810.12343*, 2018.

[4] Shashi Narayan, Shay B Cohen, and Mirella Lapata. Ranking sentences for extractive summarization with reinforcement learning. *arXiv preprint arXiv:1802.08636*, 2018.

[5] Yang Liu and Mirella Lapata. Text summarization with pretrained encoders. *arXiv preprint arXiv:1908.08345*, 2019.

[6] Ming Zhong, Pengfei Liu, Danqing Wang, Xipeng Qiu, and Xuanjing Huang. Searching for effective neural extractive summarization: What works and what's next. *arXiv preprint arXiv:1907.03491*, 2019.

[7] Yu Yan, Weizhen Qi, Yeyun Gong, Dayiheng Liu, Nan Duan, Jiusheng Chen, Ruofei Zhang, and Ming Zhou. Prophetnet: Predicting future n-gram for sequence-to-sequence pre-training. *arXiv preprint arXiv:2001.04063*, 2020.

[8] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pages 404–411, 2004.

[9] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[10] Abigail See, Peter J Liu, and Christopher D Manning. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*, 2017.

[11] Jon Deaton, Austin Jacobs, Kathleen Kenealy, and Abigail See. Transformers and pointer-generator networks for abstractive summarization.

[12] COVID-19 Open Research Dataset (CORD-19). 2020. Version 2020-MM-DD. Retrieved from https://pages.semanticscholar.org/coronavirus-research. Accessed 2020-MM-DD. doi:10.5281/zenodo.3715505.

[13] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

7 Appendix

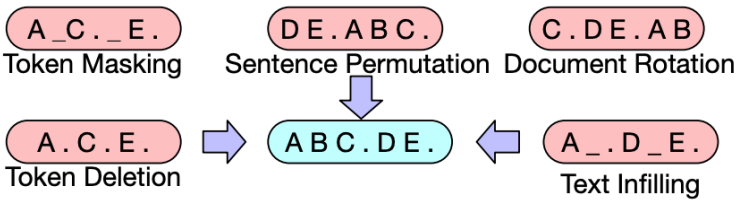


Figure 1: BART Model: Noising Strategy

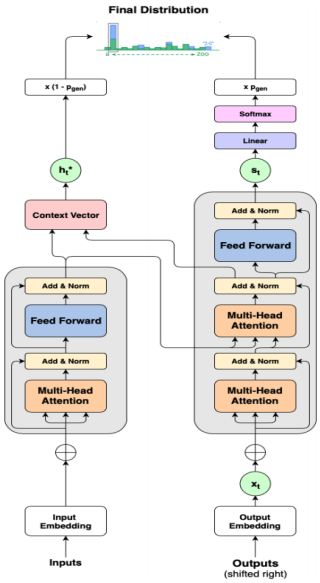


Figure 2: Transformer + Pinter-Generator

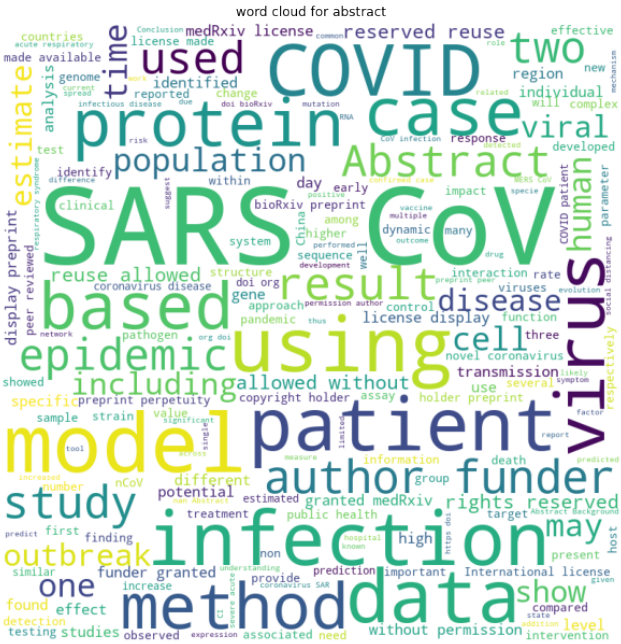
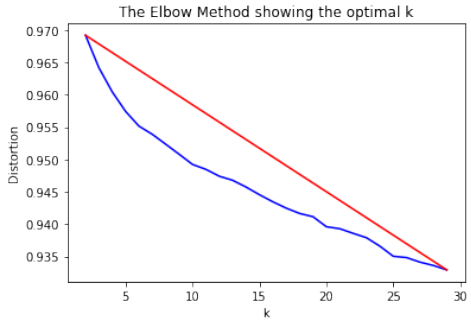
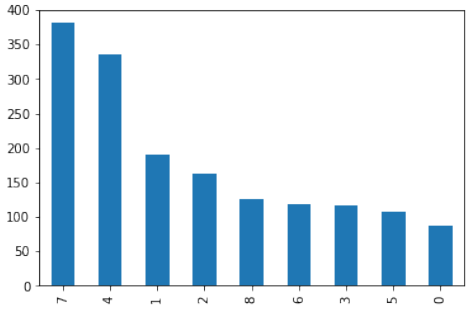


Figure 3: Word Cloud



(a) Elbow plot for choosing the number of clusters



(b) The distribution of clusters

Figure 4: Clustering Result