Text Summarization - Overview and PEGASUS Case Study

Albert Xiao, anxiao2@illinois.edu
December 2022

1. Literature Summarization

With the vast amounts of text data humans have, it becomes important to manage this data and reduce it to a digestible size. Consequently, text summarization has been a significant topic in natural language processing for the past few decades. *A Survey of the State-of-the-Art Models in Neural Abstractive Text Summarization* [1] is an overview of advances in text summarization.

1.1. Summarization Types

There are two main approaches to text summarization: extractive and abstractive. Extractive summarization involves selecting the most important sentences based on some criteria and concatenating them together. Abstractive summarization allows the generation of new words and reordering, not preserving the original text state. This makes it more similar to a human summarization. Both methods seek to generate cohesive, concise, and logical summaries.

1.2. Historical Methods

Early ideas to solve text summarization were mainly statistical approaches, using frequency and distribution metrics to determine the importance of words. The results of early methods identified the need for artificial intelligence methods to gain a more profound understanding of language semantics to achieve better results. More recent methods have appealed to this need, resulting in the current prominence of abstractive summarization. These summarization pipelines use structure, semantics, and/or deep learning to extract information, select the important pieces, and then convert them to a grammatically correct format.

1.3. Types of Language Processing Models

Within machine learning, there is a multitude of different models that have been experimented with. These include Neural Networks, Convolution Neural Networks, and Language models (Neural, Continuous Bag of Words, Skip-grams), which have evolved into state-of-the-art pre-trained models.

Currently, the most important models are sequence-to-sequence models, which involve encoding a sequence, generating a context vector, and then decoding back to text. Text summarization falls intuitively under this model, as both inputs and outputs are text.

Encoder-Decoder Models are seq2seq models that involve one encoder model and one decoder model. These can be chosen from RNN, the most basic architecture suitable for sequential data; LSTM, which is suitable for long-term dependencies; GRU, which solves LSTM's vanishing gradients issue; and bidirectional versions of RNN, LSTM, and GRU for better context finding.

Transformers are a breakthrough technique in seq2seq models, that use the attention mechanism to produce the best results. Attention is the concept that models should direct focus on pieces of the data that are more important. This is very relevant for summarization, but in general, doing so helps the model by emphasizing data segments with higher importance.

Recent research focuses on pre-trained transformers, such as BERT-GPT, BART, etc. These transformers are first pre-trained on a large text corpus using a pre-training objective, not necessarily the same as that of the given task. They are then fine-tuned for that specific task afterward. Pre-training involves large datasets, heavy computation, and long computing times, whereas fine-tuning is not as expensive. As such, this method greatly reduces training costs and pre-trained transformers can be reused for other problems. Also, fine-tuning can sometimes be done with minimal data, which helps solve novel problems with fewer data.

1.4. Mechanisms, Metrics, and Current Challenges

Researchers have employed many mechanisms to improve the results of models, including attention, copying to reuse vocabulary, pointer-generator for out-of-vocabulary and factual incorrectness problems, coverage to reduce repetition, and distraction to resolve redundancy.

Metrics used to evaluate the results of models include ROUGE and BLEU.

There are still challenges that state-of-the-art models face, which are currently being worked on. These include rare words, redundancy, long texts, factual inconsistencies, multilingual text, multi-aspect evaluation, etc.

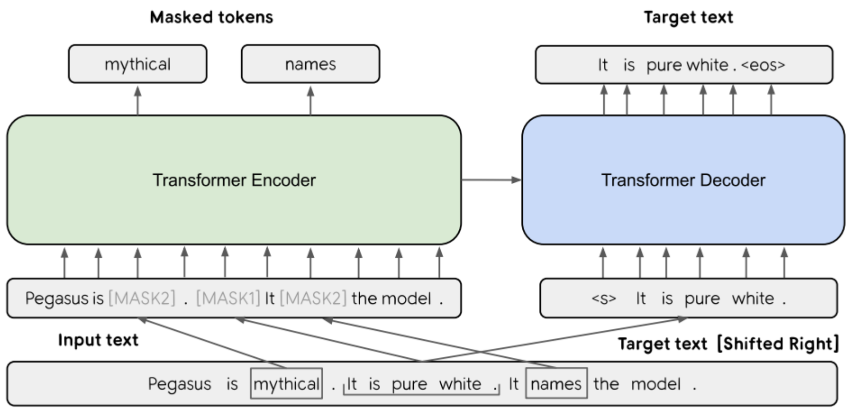
Masked tokens
mythical
names
Target text
It is pure white . <eos>
Transformer Encoder
Transformer Decoder
Pegasus is [MASK2] . [MASK1] It [MASK2] the model .
<s> It is pure white .
Input text
Target text [Shifted Right]
Pegasus is mythical . It is pure white . It names the model .

Figure 1. PEGASUS Architecture [2]

2. Model Implementation

Choosing the model to implement revolved around 2 main criteria. First, in order to generate accurate results, the model must have close to state-of-the-art performance. Second, given that I only have access to Google Colab's free GPU, the model needs to be trainable with limited GPU RAM, runtime, and computational power. The first criterion directs toward state-of-the-art transformers, such as (unironically) ALBERT, BERT, BART, GPT2, XLNET, etc. The second criterion narrowed the list down to pre-trained transformers that can be fine-tuned with low resources. Google's PEGASUS [2] architecture for abstractive summarization perfectly fits these criteria. [2] claims that PEGASUS can be fine-tuned at low resource cost, meaning relatively few data samples and optimizer steps. This is ideal for the relatively low computational resources on Google Colab.

2.1. Methodology

The PEGASUS-Large architecture (Figure 1) uses the standard transformer architecture, with 16 encoder and 16 decoder layers, hidden size = 1024, 4096 feed-forward layers, and 16 attention heads. [2] hypothesized that gap sentence generation is critical for abstractive summarization. As such, GSG is the main pre-training mechanism. This pre-training objective works by masking tokens and sentences, specifically the most important sentences determined by their ROUGE score. The model is then trained in a self-supervised manner to predict the masked sentences and words. The authors hypothesized that the GSG pre-training objective is suitable for abstractive text summarization because the model needs to not only generate (abstractive) missing text in the context of other sentences but also generate an important sentence with a high ROUGE score.

Access to the pre-trained transformer and tokenizer is provided by the HuggingFace library. The custom downstream fine-tuning training loop was developed with Pytorch with the following hyperparameters: AdaFactor optimizer, learning rate = 0.0005, data points = 1000, batch size = 1 (due to hardware RAM constraints), and epochs = 2, totaling 2000 optimizer steps. The pre-model was trained on the CNN/DailyMail dataset, one of the main summarization datasets that [2] experiments with. For full code, please see notebook.

2.2. Experiments

The main purpose of building a fine-tuning loop was to replicate the results mentioned in [2]. Specifically, I was interested in the "Zero and Low-Resource Summarization" section. In this section, the authors experimented with $10^k \, \forall k \in \{0..4\}$ training data samples, with 2000 optimizer steps and batch size 256 in order to test model performance on small fine-tuning datasets. To simulate this experiment, I evaluated my model after being fine-tuned with 0 samples (zero-resource) and 1000 samples (low-resource). Similar to [2], ROUGE-1, ROUGE-2, and ROUGE-F metrics were used as evaluation criteria. These scores were calculated from the results of generated summaries of 500 samples from each of the CNN/DailyMail and XSum test sets. Further texts from the internet were taken for human evaluation.

2.3. Discussion

Comparing my results with [2] in Table 1, it is clear and expected that my results are significantly worse in all three ROUGE scores. This is obvious from the fact that [2]'s model is fine-tuned on the full dataset, with a much larger batch size and more epochs. On the other hand, my model's results are realistically comparable with the zero and low resources results of [2].

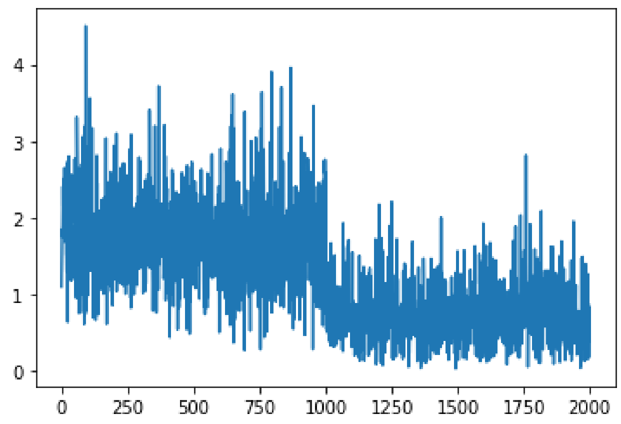

Figure 2. Fine-tuning Loss

Test Dataset	Model	ROUGE1	ROUGE2	ROUGEL
CNN/DailyMail	Mine	30.13	11.31	21.76
CNN/DailyMail	Google	**44.17**	**21.47**	**41.11**
XSum	Mine	21.24	5.13	15.21
XSum	Google	**47.21**	**24.56**	**39.25**

Table 1. My Model Performance after 2 Epochs of fine-tuning on 1k data VS Paper results after full fine-tuning, better scores are in bold

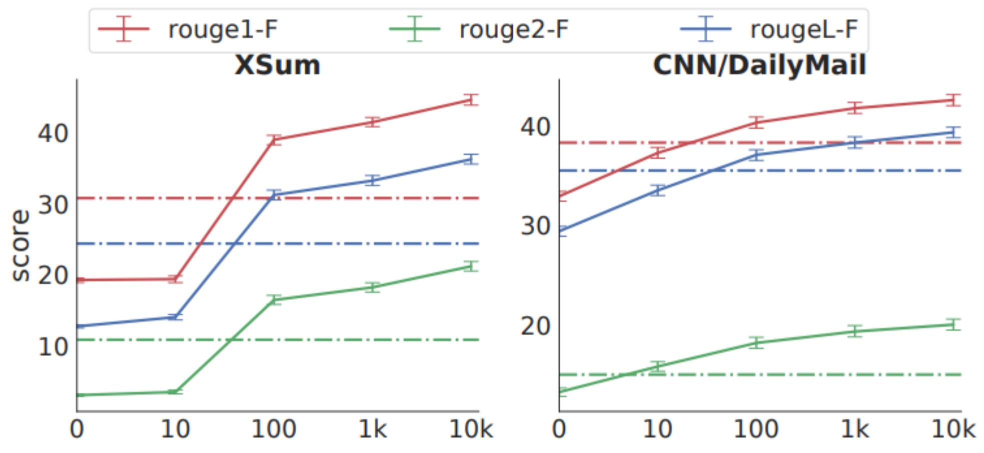

Figure 3. Low resource experiment results [2]

Test Dataset	Data	ROUGE1	ROUGE2	ROUGEL
CNN/DailyMail	1000	**30.13**	**11.31**	**21.76**
CNN/DailyMail	0	25.00	8.40	16.84
XSum	1000	**21.24**	**5.13**	**15.21**
XSum	0	16.51	2.61	11.23

Table 2. My Model Performance after 2 Epochs of fine-tuning on 1k data VS no fine-tuning (0 data), better scores are in bold

First, only looking at my own model's experimental results in Table 2, it is notable that for both datasets, all three ROUGE scores experienced a somewhat significant increase after fine-tuning. Such an observation is desirable, as it shows that fine-tuning had a large positive effect on scoring metrics despite being low-resource. We also see that the XSum dataset has lower scores than CNN/DailyMail, which is also expected because the model was fine-tuned on CNN/DailyMail. The idea that fine-tuning positively affected the results is also supported by the training loss graph in Figure 2. We can see a visible reduction in loss after 2000 iterations, suggesting that the model became more accurate. Lower loss due to overfitting is unlikely due to the high parameter count of the model.

We can also compare my model's results to the results of [2]'s fine-tuning on zero and low resources shown in Figure 3. In this figure, the x-axis is the number of data samples used in fine-tuning. Since I only experimented with 0 and 1000 samples, we will focus on the ROUGE scores for $x = 0, 1000$. Compared to the paper's results, we see that our zero-resource (0 data) results are similar, but my low-resource (1000 data) results are worse. Again, this is attributed to the smaller batch size (hardware constraints) while having the same number of optimizer steps. In the future, it would be interesting to train my model to a more equivalent state and compare results (unfortunately Google Colab limits free GPU hours). Despite less appealing results, the increase in ROUGE scores after low-resource fine-tuning corroborates the claim that PEGASUS performs well with low resources.

2.4. Human Evaluation

My results are easier to interpret with human evaluation (see Appendix A for full texts and summaries). For this section, I selected three pieces of text: a section of a CNN article, a Wikipedia article, and *The Adventures of Sherlock Holmes*. I was curious about model performance on different types of text and writing styles. One interesting observation is that the zero-resource results are fully extractive, not generating any new text. This makes sense because the pre-training object is indeed extractive as it encourages the model to regenerate the masked text exactly rather than generate new text. Contrastingly, we see the abstractive features in the low resource results, as there are many paraphrased ideas. Furthermore, the summaries generated for these three excerpts do seem to be cohesive, non-redundant, grammatically correct, and logical, which I would say is fairly successful despite lower ROUGE scores. This is especially interesting because the model was able to "understand" the complex biology concepts in the Wikipedia article. It also summarized the fictional text well despite being fine-tuned on non-fiction data.

3. Chrome Extension

3.1. Methodology

The chrome extension I built uses an HTML/CSS/JS frontend using the chrome API, connected to a Django backend, which runs the model and serves the data. A button press on the popup sends a request to the content script, which passes the request along with the selected or all text on the webpage to the background script. The background script makes an HTTP post request to the Django API. Then the API, which has PEGASUS pre-loaded, generates the summary and sends the summary back up the chain to the popup, where it is displayed. See Figure 4 for a visual pipeline. Often, on webpages, there are frequent advertisements, menus/headers, and other textual noise on the screen. So, I have added further functionality by allowing the user to select a portion of text with the cursor, and the generated summary will only contain content from the selected text.

3.2. Improvements

The current version of the chrome extension is quite good for small segments of text. The source of the text does not have to be news articles. Wikipedia pages and fictional passages work as well. However, long text inputs are a difficult challenge. PEGASUS only supports tokenizing up to 1024 characters. Moreover, tokenizing more characters will likely result in poor results as it is not trained on long text. To resolve this problem, a workaround is to segment the text into chunks of 1024 tokens and concatenate the results for each chunk. Because there is no memory or context between chunks, an issue with this method is that summaries can be redundant, as the model does not remember previously summarized points and concepts. Thus, models with larger tokenization capabilities are required. Longformer [3], which has been trained on long legal documents and supports up to 4096 tokens, and GPT3 [4] are possible candidates.

4. Conclusion

4.1. Applications

Text summarization can be used to create a comprehensive studying tool for students. For this, students can input study material, such as lecture slides, and textbook sections. The summarizer would then scrape, parse and identify key concepts for the student. It would also generate short summaries for each concept to help the students get a big-picture understanding. Another functionality would be that the student could select a topic, and the software would filter and summarize content pertaining only to that topic. Another interesting idea would be to add text summarization to augmented reality glasses in conjunction with computer vision. As there is a lot going on in the world around us, we often miss a lot of information. For this idea, the computer vision portion would identify important objects, and translate their states to text, while another computer vision module would collect surrounding visible text. After combining these two portions, the summarization en-

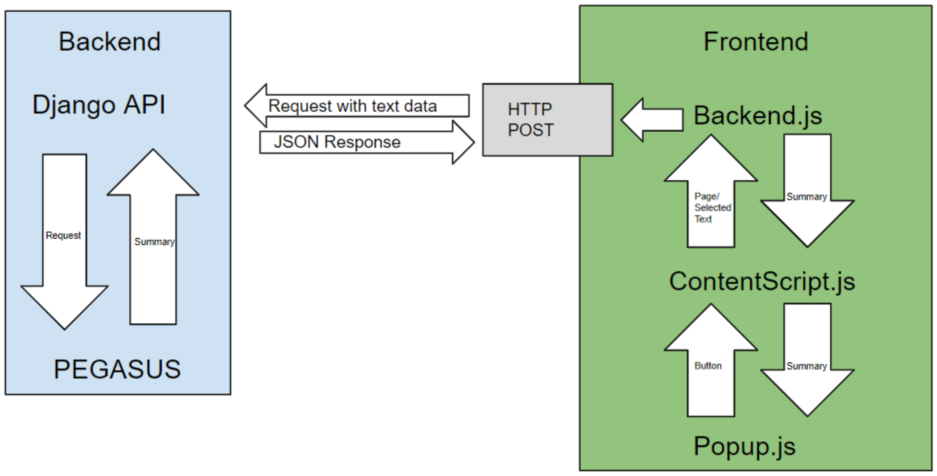

Figure 4. Chrome extension pipeline

gine would then display the relevant and important sections on the AR glasses so that the user can be more aware of their surroundings.

4.2. Learning Reflections

I gained a lot of insights and knowledge from this assignment. From reading the survey paper, as well as several referenced papers as supplementation, I now have a much better understanding of state-of-the-art language models. These papers reinforce my understanding of concepts such as encoder-decoder models while providing me with new information, such as pre-training and fine-tuning mechanisms. Furthermore, I've witnessed many brilliant mechanisms and tricks papers use to improve results, such as using a pre-training objective not specifically for the problem but is similar enough in a clever way to still produce good results. Seeing these ideas encourages me to approach problems from multiple angles, even unexpected ones. Through the project, I have also witnessed that there are still challenges to current techniques, and there is still room for improvement. This makes me motivated to take on these challenges. The project has definitely enhanced my ability in scientific experimentation, academic writing, and development. Overall, this 'crash course' and project with NLP have definitely broadened my horizons, and I am eager to take a deep dive into this field.

5. References

[1] A. A. Syed, F. L. Gaol and T. Matsuo, "A Survey of the State-of-the-Art Models in Neural Abstractive Text Summarization," in IEEE Access, vol. 9, pp. 13248-13265, 2021, DOI: 10.1109/ACCESS.2021.3052783.

[2] Zhang, Jingqing, et al. "Pegasus: Pre-training with extracted gap-sentences for abstractive summarization." International Conference on Machine Learning. PMLR, 2020.

[3] Beltagy, Iz, Matthew E. Peters, and Arman Cohan. "Longformer: The long-document transformer." arXiv preprint arXiv:2004.05150 (2020).

[4] Brown, Tom, et al. "Language models are few-shot learners." Advances in neural information processing systems 33 (2020): 1877-1901.