Summarization of News Articles Using PEGASUS

Gregory LeMasurier
University of Massachusetts Lowell
Gregory_LeMasurier@student.uml.edu

Mojtaba Talaei Khoei
University of Massachusetts Lowell
Mojtaba_TalaeiKhoei@student.uml.edu

Abstract

Summarizing large bodies of text is necessary so that readers can get a high level understanding of an article, without requiring them to read the entire body of text. While authors can write their own summaries, automating this process would save them time. In this work we compare a base line randomly initialized PEGASUS model to a pretrained PEGASUS model that was fine-tuned on the cnn_dailymail dataset.While our models are undertrained, they show acceptable performance given their rouge scores. Additionally we show the generalizability of our fine-tuned model on three different article types. Finally, we offer potential ways to improve our models' performance in future work.

1 Introduction

Summarizing large bodies of text is beneficial to enable readers to quickly understand the main points of the text. For example, by reading the summary of a news article a reader can understand the described events at a high level and if they would like to get a deeper understanding of the news event they can then decide to read the whole text. Additionally summarization is beneficial in academic work. Academic papers include an abstract section that provides readers with a summary of the contributions of the work. The abstract enables the reader to determine if the paper is relevant to their own work so they can decide if they should continue reading the lengthy full paper. While many summaries are generated by the authors of the article, automating this process can save the authors time, and would additionally enable readers to summarize long articles which do not include summaries. One subfield of Natural Language Processing (NLP) focuses on text summarization.

There are two dominant strategies for text summarization in NLP. The first approach is extractive summarization which aims to summarize text by taking sentences from the original text. These approaches determine how important a particular sentence is to the overall article, where the most important sentences are added to the summary. While it is valid to extract full sentences to produce a summary, the produced summaries are not as concise as they can be since not all information in a sentence is necessary to include.

The second approach to summarization, abstractive summarization, was designed to solve this problem. In abstractive summarization, the model learns the important sentences and their context in the overall article. These important sentences are then rephrased to generate more concise summaries.

One state of the art method for abstractive summarization is PEGASUS (Zhang et al., 2020). In this work, we investigate fine-tuning a pretrained PEGASUS model on the cnn_dailymail dataset (Hermann et al., 2015). We then compare the performance of this model to a randomly initialized PEGASUS model. Next, we show that our model can produce summaries for a variety of types of articles, showing the generalization of this approach from news articles to other sources of media. Finally, we discuss limitations with our work which could be addressed in future work.

2 Related Work

There are many different approaches to abstractive summarization. One state of the art approach uses a generalizable pretraining framework that they call General Language Model (GLM) (Du et al., 2021). This approach utilizes positional encodings to improve their blank filling pretraining. Additionally, they enable their model to predict spans in a random order. The GLM-XXLarge model currently is ranked as the top model for abstractive summarization on the cnn_dailymail dataset[1].

[1] `https://paperswithcode.com/sota/abstractive-text-summarization-on-cnn-daily`

One popular framework for summarization is BART (Lewis et al., 2019). BART is a transformer based denoising autoencoder that is commonly used to pretrain sequence-to-sequence models. BART first applies a noising function to the input text, and then learns to convert the noisy data back to the original text. Wu et al. applied a regularized dropout to BART to improve performance (Wu et al., 2021). Additionally Aghajanyan et al. found that their prefinetuning method applied to BART outperformed their summmarization baselines (Aghajanyan et al., 2021).

Additionally, groups have investigated summarization by fine-tuning models using trust-region theory (Aghajanyan et al., 2020). This fine-tuning method performed better as it solves representational collapse, where representations degrade during fine-tuning ultimately producing a less generalizable model.

Another state of the art approach is PEGASUS (Zhang et al., 2020). PEGASUS is a standard sequence to sequence transformer encoder-decoder model. PEGASUS was pretrained on masked language modeling and gap sentences generation objectives using the C4 and HugeNews datasets. The PEGASUS model was found to achieve state of the art performance on small datasets with few samples. In this work we explore fine-tuning and training a pretrained PEGASUS model compared to a randomly initialized PEGASUS model. We additionally compare our results to the state of the art methods mentioned above in Section 4.

3 Methodology

In this work, we focus on abstractive summarization using PEGASUS (Zhang et al., 2020). We created a randomly initialized PEGASUS model for a base line performance and then fine-tuned a pretrained PEGASUS model. A high level overview of the PEGASUS model can be seen in Figure 1. First the article text is tokenized using a tokenizer. The tokenized text is then sent to the PEGASUS model. The PEGASUS model is a standard seq to seq transformer encoder decoder model, which uses 16 encoder and 16 decoder layers that are connected by a context vector. PEGASUS models differ from standard approaches as they are pretrained on masked language modeling and gap sentences generation objectives. Additionally, the PEGASUS models proposed by Zhang et al. were trained on the C4 and HugeNews datasets (Zhang et al., 2020). The tokenized article text was passed as input ids to our model. Additionally we passed in the attention mask which indicated which tokens were padding tokens to the model. The model also took in labels, which it used to calculate the loss. The PEGASUS model output a set of logits. These logits can be sampled using Top-K sampling. In Top-K sampling, we select the top K logits to use in the summary. We initially used Top-K sampling to calculate the loss in our pretrained model. This was not necessary in the final version of our pretrained model as the loss was automatically calculated by the model. According to huggingface documentation, the PEGASUS models by default use a beam size of 8 to sample and create the summary.

3.1 Dataset

Our models are fine-tuned and trained on the cnn_dailymail dataset (Hermann et al., 2015). This dataset consists of over 300k unique news articles from cnn and dailymail that are written in English. Each data entry consists of an identification number, an article, and a highlight, or summary, of the article that was written by the author of the article. The mean token size is 781 for articles and 56 for highlights. This dataset consists of 287,113 samples for training, 13,368 for validation, and 11,490 for test. Our validation step was taking roughly one hour to complete, so we randomly sampled 3000 entries from the validation set to use in our training loop.

3.1.1 Tokenizer

The tokenizer used in this work was the google/pegasus-xsum tokenizer. This tokenizer has a vocab size of 96103. When tokenizing the dataset we used a DataCollatorForSeq2Seq collator which would truncate our text to the maximum sequence length of 512. While the mean article size was 781 tokens, we had to limit our sequence length to 512 to fit in the GPU memory. We expect that this will lower our scores as we do not use the full article to produce the summary. If the article was shorter than 512 tokens it would add padding tokens. The PEGASUS tokenizers using a padding token with id 0 rather than the default -100 token id.

3.2 Base Model

A base line model is necessary to compare the effectiveness of our fine-tuned model. Thus, we create a a base PEGASUS model with randomly initialized weights. This PegasusForConditionalGeneration

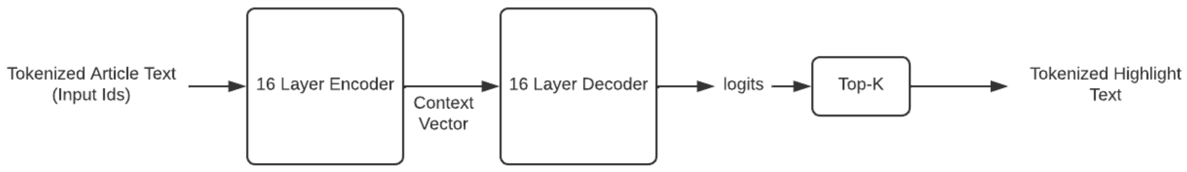

Figure 1: A high level overview of the PEGASUS model.

model was created using a base PegasusConfig. The PegasusConfig was created with use_cache set to false. This helped reduce the total GPU memory consumption as the model would not output the last key and value attentions, which were not used by our algorithm. Additionally, we set max_position_embeddings to be equal to the maximum sequence length we could fit on the GPU, which was 512. The vocab_size used in this config was set equal to the vocab_size of the tokenizer, which was equal to 96103.

Since this randomly initialized model was created as a base line, we did not change the other default parameters. This base model uses a total of 12 encoder layers and 12 decoder layers. Additionally, it uses 16 encoder and 16 decoder attention heads. The encoder and decoder layers use a feed forward dimension of 4096. Finally, a dropout of 0.1 was used in this base model. This model was trained on Google Cloud's A100 GPU for 3 epochs.

3.3 Pretrained Model

To load our pretrained model, we created a PegasusForConditionalGeneration from the google/pegasus-xsum pretrained model. This model was trained on the C4 and HugeNews datasets. The pretrained model was fine-tuned on the xsum dataset. This model uses a total of 16 encoder layers and 16 decoder layers. Like the base model, the pretrained model also uses 16 encoder and 16 decoder attention heads. The encoder and decoder layers also use a feed forward dimension of 4096. Similarly to the base model, a dropout of 0.1 was used in this pretrained model. The pretrained model was fine-tuned on Google Cloud's A100 GPU for 2 epochs.

3.4 Training Loop

In our training loop, we used a learning rate scheduler which started with a learning rate of 5e^{-5}. Every 30,000 steps, we would preform an evaluation of the Rouge-1, Rouge-2, Rouge-L, and Rouge-LSum scores. We would do so using the 3000 randomly sampled validation samples as using all 13,368 samples would take too long to process, ultimately also being too expensive as it would add one hour to our training time for each evaluation. The largest challenge we faced while training and fine-tuning our models was GPU memory consumption. One potential way to reduce GPU memory consumption would be to lower the maximum sequence length. This is not ideal for summarization as we need as much of the original article text as possible to generate accurate summaries. Thus, we tried to tune our other parameters to increase our sequence length to be as large as possible. To do so, we used a smaller batch size of eight with four gradient accumulation steps to effectively have a batch size of 32. Additionally, used the Adam8 optimizer (Dettmers et al., 2021), rather than the standard Adam optimizer, to reduce memory consumption. After completing the training loop, we would then evaluate our model on the entire test set.

4 Results

To evaluate our models we used the Rouge-1, Rouge-2, Rouge-L, Rouge-LSum metrics. Figure 2 shows the models' Rouge scores throughout training. These graphs show that our models are fairly under trained. This is shown as our f1 scores are not decreasing as the model trains. As expected, our pretrained model has a much higher Rouge score than the randomly initialized model. This is expected in the pretrained model, we use the pretrained weights as a starting point, where as we start with random values in the base model.

Additionally, we compared our models' performance to the state of the art models. Table 1 shows our models performance compared to the state of the art models for abstractive summarization. The state of the art models were selected by their ranking on paperswithcode for abstractive summarization on the cnn_dailymail dataset[2]. As seen in the

[2]https://paperswithcode.com/sota/

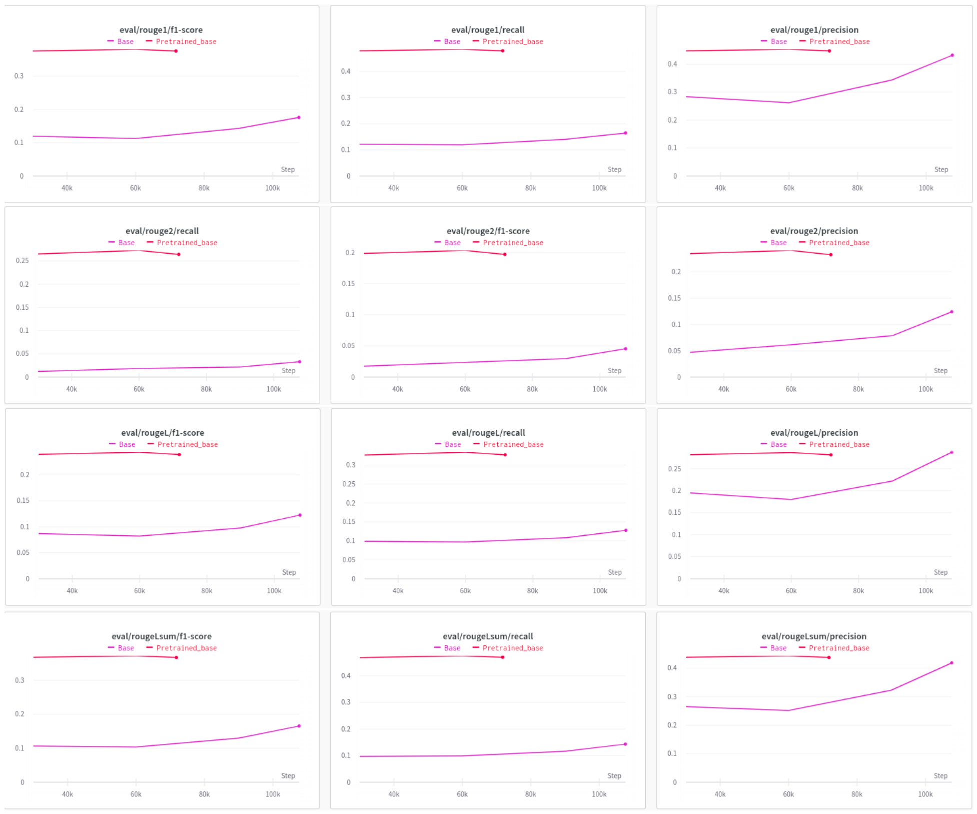

Figure 2: The Rouge scores of our base model (pink) and pretrained model (red) during training.

table, our fine-tuned model performed alright compared to the state of the art models. Since our models were under trained, if we had trained them for longer we expect that we would have gotten results more comparable to the PEGASUS LARGE model, the best model proposed by Zhang et al. (Zhang et al., 2020).

Since our pretrained model was pretrained on news articles and then fine-tuned on other news articles, we investigated how our model performed on various types of text. First we tried a news article[3], the output summary was:

"The decision to keep more water in Lake Powell on the Arizona-Utah border comes as both are at record-low levels after 20 years of drought made worse by climate change . The decision will keep the lake at a level at which it can continue generating hydropower for the next 12 months .".

Next we tried a random wikipedia page[4], which was summarized to:

"The Ares V was to launch the Earth Departure Stage and Altair lunar lander for NASA's return to the Moon, which was planned for 2019 . The uncrewed Ares V would complement the smaller, and human-rated Ares I for the launching of the 4–6 person Orion spacecraft".

Finally, our model summarized academic text as seen by the summary generated of the abstract of the PEGASUS paper (Zhang et al., 2020):

"We propose pre-training Transformer-based encoder-decoder models on massive text corpora with a new self-supervised objective. We evaluated our best PEGASUS model on 12 downstream summarization tasks spanning news, science, stories, instructions, emails, patents, and legislative bills. Experiments demonstrate it achieves state".

This shows that our model is capable of adapting to various types of articles. Thus, our fine-tuned pretrained model is generalizable to various different types of text, not just news articles.

5 Discussion

Ultimately, through this project we learned about different forms of text summarization and the state of the art approaches to this NLP problem. We also

abstractive-text-summarization-on-cnn-daily

[3] https://www.nytimes.com/2022/05/03/climate/lake-powell-mead-water-drought.html

[4] https://en.wikipedia.org/wiki/Ares_V

Table 1: Table comparing Rouge-1, Rouge-2, and Rouge-L scores of the state of the art models for abstractive summarization. The performance of our models can be seen in bold.

Model Name	**Rouge-1**	**Rouge-2**	**Rouge-L**
GLM-XXLarge (Du et al., 2021)	44.7	21.4	41.4
BART + R-Drop (Wu et al., 2021)	44.51	21.58	41.24
MUPPET BART Large (Aghajanyan et al., 2021)	44.45	21.25	41.4
BART+R3F (Aghajanyan et al., 2020)	44.38	21.53	41.17
ERNIE-GENLARGE (Xiao et al., 2020)	44.31	21.35	41.60
PALM (Bi et al., 2020)	44.30	21.12	41.41
ProphetNet (Qi et al., 2020)	44.20	21.17	41.30
PEGASUS LARGE (Zhang et al., 2020)	44.17	21.47	41.11
Fine-Tuned PEGASUS	**36.7**	**26.42**	**23.91**
Random Base	**18.52**	**4.33**	**12.57**

got more hands on experience creating a randomly initialized state of the art summarization model, as well as experience using a pretrained model that was proposed in an academic paper.

Through this process, we learned about gradient accumulation and tried various ways to conserve GPU memory consumption. We also learned about and got experience using the Adam8 optimizer. Additionally, we used Rouge scores which were new metrics to us.

While we did get acceptable results, we believe that we could further improve our models by improving the efficiency of our validation step. This would save us a lot of time and would enable us to train and fine-tune our models for longer. This is necessary as our models were under trained, training our models for longer would certainly increase their performance.

Finally, our models' performance could be improved if we could further reduce GPU memory consumption. We had to limit our sequence length to 512 tokens, which is roughly 200 tokens shorter than the average article length. If we could increase our sequence length to 1024 tokens, our model could work with more of the original article, resulting in more accurate summaries.

References

Armen Aghajanyan, Anchit Gupta, Akshat Shrivastava, Xilun Chen, Luke Zettlemoyer, and Sonal Gupta. 2021. Muppet: Massive multi-task representations with pre-finetuning. *arXiv preprint arXiv:2101.11038*.

Armen Aghajanyan, Akshat Shrivastava, Anchit Gupta, Naman Goyal, Luke Zettlemoyer, and Sonal Gupta. 2020. Better fine-tuning by reducing representational collapse. *arXiv preprint arXiv:2008.03156*.

Bin Bi, Chenliang Li, Chen Wu, Ming Yan, Wei Wang, Songfang Huang, Fei Huang, and Luo Si. 2020. Palm: Pre-training an autoencoding&autoregressive language model for context-conditioned generation. *arXiv preprint arXiv:2004.07159*.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 2021. 8-bit optimizers via block-wise quantization.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2021. Glm: General language model pretraining with autoregressive blank infilling.

Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *NIPS*, pages 1693–1701.

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

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

Lijun Wu, Juntao Li, Yue Wang, Qi Meng, Tao Qin, Wei Chen, Min Zhang, Tie-Yan Liu, et al. 2021. R-drop: regularized dropout for neural networks. *Advances in Neural Information Processing Systems*, 34.

Dongling Xiao, Han Zhang, Yukun Li, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. 2020. Ernie-gen: An enhanced multi-flow pre-training and fine-tuning framework for natural language generation. *arXiv preprint arXiv:2001.11314*.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. 2020. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pages 11328–11339. PMLR.

A Appendix

Contributions:
As a team we would meet in person and sometimes online to work on the project. We simultaneously worked on processing the dataset, working with the tokenizer, and setting up the training loop together in person. We did this to prevent merge conflicts that would arise if we worked on these core aspects of the project separately. We each worked on loading a single model, the randomly initialized base and the pretrained model. Both of these required the same amount of work. Additionally, we also worked on the presentation and demo together in person. We both collaborated on this paper online.