Academia Summarized

Aditya Duddikunta
aduddikunta3@gatech.edu

Scott Goldstein
sgoldstein40@gatech.edu

Joseph Hirschi
jhirschi3@gatech.edu

Rajneesh Kumar
rkumar319@gatech.edu

Abstract

Transformer based encoder-decoder models are currently the state of the art on text summarization tasks. Although a lot of work has been done on their pretraining, research on how to improve fine tuning for generating credible abstract summarization of the text from highly specialized domains can use more exploration. In this work, taking a wide variety of technical papers in STEM, we show that different parts of the paper have varying levels of effectiveness when it comes to fine tuning and our findings are consistent with our intuitive expectations. We also demonstrate that fine tuning, preceded by an extractive step for important sentences, rather than random selection of training text, is more effective; across different test samples and paper types. Finally, by combining the idea of the suitable section with an extractive step, we achieve a better ROUGE-L [3] score than the current baseline on the abstractive text summarization task.

1. Introduction/Background/Motivation

This paper focuses on automatic text abstractive summarization, the process of creating concise summaries of content that retain the meaning and most important information. This is different from extractive summarization which aims to find the principal sentences that best capture the idea of a piece of text. Unlike its extractive counterpart, abstractization involves creating new sentences that accurately and succinctly capture the gist of a piece of writing. Imagine staring at a 100-page report with the task of succinctly explaining the main points to a person of importance in the next 30 minutes. Now imagine a similar assignment tasked monthly, weekly, or even daily. How could that possibly be done accurately? Automatic text summarization is of interest in the deep learning community because it's a stimulating task with the potential of positively impact a wide-reaching set of industries and organizations. This work aims to contribute to the research by experimenting with ways to improve the current state of the art (SOTA) a popular text summarization model, [10, 11].

In some industries, text summarization is still done manually as adoption to automatic methods has been slow. However, with automated text summarization, there have been a number of techniques developed recently, most of them based on extractive summarization. In abstractive summarization, which generates new sentences as opposed to just copying information, sequence-to-sequence models using encoder-decoder architectures based on RNN's and Transformers haves become a popular framework [8]. PEGASUS, the baseline model this paper aims to improve upon, is the current SOTA architecture [11]. PEGASUS is another effective text summarization model that pre-trains a large Transformer-based encoder-decoder architecture on text with a new self-supervised objective [11]. In PEGASUS, important sentences are masked from an input textpaper and are generated together as one output sequence from the remaining sentences [11]. Currently, the main limitations are around the extent of abstractization as well as the size of the input and output text that can be handled by these models [8]. Moreover, the performance may depend on the domain of the text and may have trouble generating effective summaries for texts from specialized domains. We encountered these limitations first hand while working with technical STEM papers on this project [8].

As previously mentioned, accurate text summarization could positively impact many industries by changing the way work is done and improving performance. For instance, law firms are constantly reviewing documents that are hundreds of pages long and accurate summaries could enable attorneys to quickly understand key points and better focus their attention, which results in substantial time and cost savings. Another example involves national security and intelligence collection. With accurate text summarization, analysts, tasked with distilling massive amounts of text information into concise reports for policy makers, could sift through documents quicker and identify the relevant information. This could result in an increase in the number and quality of reports that help drive policy. These are

merely two examples of how improved text summarization could improve outcomes and there are many more across industries such as academia and law enforcement. Abstractive summarization will also push the frontier of NLP research and help the researchers come up with better architectures that can overcome the current limitations associated with Transformers (e.g. lack of semantic knowledge of a language and inferential reasoning) [8].

Transformers-based architectures have a number of limitations that make them a challenge to use [8, 10]. The first is that input tokens are limited to around 1000 tokens for most architectures which makes this a challenge when trying to perform a task on a longer piece of text [10]. Most approaches to this problem truncate the text when tokenizing. However, this can cause important content to be missing from the input if important information is present in the middle to end of the paper. Another common limitation of transformers is their memory hungry nature. Transformers have shown to capture and store the syntactic information well but lack the capability of commonsense inference and reasoning [8]. It has been widely reported that attention heads in the same layer tend to learn the same pattern which points to a need for improvement in the design [8]. It is also suspected that Transformers tend to learn shallow heuristics due to shortcomings in the current optimization methods.

In this paper, experiments were performed using the Kaggle arXiv dataset, which consists of over 1.7 million scholarly papers across STEM [9]. Cornell University operates the dataset and set up a free Google Storage bucket to make it accessible [9]. The dataset includes features such as article titles, authors, categories, abstracts, and the original PDF of the paper. The full text of each paper was extracted from the PDF using a standard PDF parser called PyMUPDF and the sections of the text were extracted using a parser from AllenAI that is specifically tuned to extract sections and headers of scientific papers [2, 7]. In total, given our resource constraints, it was impossible to utilize the full dataset of 1.7 million papers, and therefore, we extracted 14,000 texts out of the 1.7 million texts to use as our dataset.

2. Approach

Our goal was simply to output summaries of academic papers given a specific piece of input text. Given this problem, it is readily apparent that this is a generative conditional language task in that our approach has to be to generate a sequence conditioned upon an input sequence that we feed in. In our case, we took the abstract of a paper as a representative summary of a paper, and we conditioned our model on the text of the paper. The current SOTA for these classes of problems are transformer-based language models which are suitable for a wide variety of natural language problems including the problem of summarization [8]. The model that we will use for the rest of this paper is the PEGASUS-large model from Google AI. Pegasus's pretraining task is to generate a sequence based on a masked input which is somewhat similar to the downstream task of summarization [11].

To produce an initial baseline, we first tried to produce summaries using the model to predict without any sort of fine-tuning or additional training on our input texts. Given this baseline, we then attempted to improve upon our baseline by trialing a variety of techniques. The first was to finetune using the whole input text. This was problematic since the PEGASUS model has a limit on the number of input tokens so when tokenizing the entire body of the paper, it was inevitable that sections would essentially be cut off due to a truncation operation [10]. Given this problem, we then attempted to only train on specific subsets of the paper in order to sidestep the limitation on the number of input tokens. For instance, we finetuned the model on only the introduction or only the conclusion/discussion sections of the paper. Generally speaking, fine-tuning on select inputs rather than fine-tuning on the whole text appeared to produce greater predictive performance.

We further extended the process of selective fine-tuning by adding an extractive layer prior to the summarization layer [5, 6]. Instead of heuristically grabbing certain sections, we instead used a separate BERT transformer to grab the N most important sentences from the paper's body and then feed that output into the PEGASUS model to produce a summarizer. Instead of solely grabbing text from one section of the paper, we are able to selectively grab the most important sentences and then have our summarization layer produce a summary conditioned upon these extracted sentences [5, 6]. This approach allows greater focus on the most important sections of the paper while also still being able to make use of every part of the input text [5, 6]. In addition, we experimented with using an extractive layer on select sections of the text instead of the full input text. Both approaches yielded better results than naively grabbing individual sections of the paper.

We then proceeded to go further in an attempt to reduce the resource footprint of the training process by performing extraction on both the input and the ground-truth label, the abstract of the text. This essentially creates a proxy task for the transformer to learn rather than the real task of learning the full abstract of the text given an input. We hoped that the network would still be able to learn the necessary context to be able to perform well on the downstream task which is predicting the full abstract. This reframing of the training task reduced the GPU memory usage to a level that allows one to perform this task on a 16 GB GPU which has a more digestible memory footprint than from before when training using full abstracts.

Throughout the training process, memory usage was a

Finetune (Y/N)	Recall	Precision	F-Score	GPU Usage
N	0.1212	0.4823	0.1938	5 GB
Y	0.3559	0.2599	0.2594	48 GB

Table 1. Baseline Model Performance vs Finetuned Model

persistent concern due to our limitations in compute and time. Our first experiments frequently failed due to the extremely high memory requirements that the full text required. This is a main drawback of transformer-related approaches since they frequently require large GPU's that are in excess of 40GB. One of the ways we counteracted this problem was simply to train using larger GPUs essentially bypassing the problem by using more computational power. However, this is not an ideal solution, so we improved upon this situation by attempting to train with smaller and smaller extracts of both the input and label and testing whether at inference time, the performance by the finetuned model was still satisfactory.

3. Experiments and Results

Our main goal was to use Transformer-based architectures to summarize our corpus of academic texts. The first measure of success was whether we could beat the baseline performance of PEGASUS by fine-tuning the model on our corpus of text. If we could beat the baseline performance of PEGASUS, we considered that as a successful outcome. We then wanted to see if we could improve upon that performance with varying modifications to our original experiments in the form of modifying input or doing some additional preprocessing step on the input. If any of these methods yielded a greater performance than both the finetuned performance as well as the PEGASUS baseline, we considered that as a successful outcome. Along the way, we also wanted to evaluate the "abstractness" of our summarization and make sure that our models were not simply copying swathes of text from the original and rearranging them into a different order. There was no easy way to systematically determine this automatically, so part of our evaluation was to do this qualitatively. Finally, given the expected memory constraints that we ran into, we sought ways to decrease our memory usage and still maintain satisfactory performance on the test set.

3.1. Scoring Metrics

The rouge-L scoring metric is a particular metric that is useful in evaluating how close a candidate summary is to the reference summary [3]. In particular, ROUGE-L scores at a summary level calculate the longest common subsequence between every pair of reference/candidate sentences and computes what is essentially a union operation on the

Section	Precision	Recall	F-Score	GPU Memory Usage
Intro	0.3665	0.2646	0.2844	32 GB
Middle	0.4457	0.2332	0.3062	44 GB
Disc./Conc.	0.5253	0.2540	0.3424	36 GB
Full Text	0.3559	0.2599	0.2594	48 GB
Untuned	0.3559	0.2599	0.2594	5 GB

Table 2. Model Performance on Subsets of Text

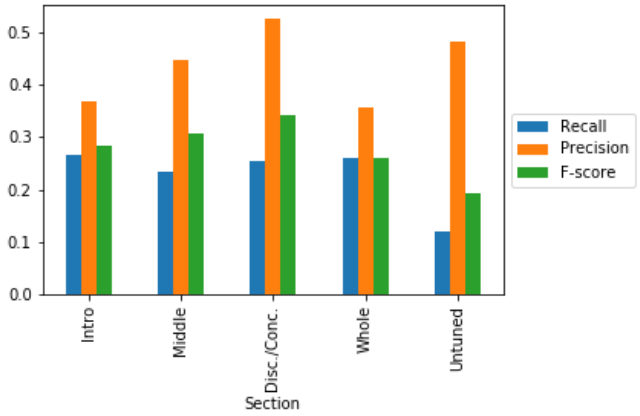

Figure 1. Fine-tuned Model Performance

result of this [3]. This is the standard way in the literature to compare two summaries and is therefore what we have decided to use as a metric [3, 11].

In addition, we inspected several summaries and evaluated them manually. This may seem a somewhat primitive way of assessing a summary's quality, however, many of the benchmarks task in the area of summarization now involve human input and human perception about the quality of a summary [11]. Therefore, we believe that substantive qualitative discussions can be had by inspecting the contents of a summary and comparing to our ground-truth label, the abstract.

3.2. Baseline Models

Our first set of experiments was simply to benchmark the current existing approaches on our downstream dataset. We can see the performance of our untuned PEGASUS models on the full input text in Table 1. We can see that precision is high, however, recall is lagging behind meaning that the predicted summary is not capturing enough of the reference's content, and therefore the F-score is low. Given this initial benchmark, we then finetuned the PEGASUS model on our dataset of papers using the full text of the paper as an input which yielded the following results. We can see clear improvements in recall and the F-score indicating that the model is trading off precision in order to increase recall. This essentially means that the sentences

being outputted are more inaccurate when compared to the reference abstract, however, we have captured more of the abstracts' contents. The increase in F-score indicates that finetuning the model on our dataset seemed to increase the performance of the model on this dataset.

3.3. Training on Sections vs Baseline

Given the baseline, we then looked to beat that performance by training a model on only subsets of our input. In Table 2 and Figure 1, we see the results of training on these subsets. We can see that the greatest performance by F-score is among the class of models where the model is trained on the discussions/conclusions section followed by the middle of the text and then by the introduction. This is possibly because selecting certain sections of the text has the effect of focusing the learning algorithm on the important parts of the paper for summarization. For instance, in many cases, a human can summarize an academic paper by solely looking at the discussion section because that is where the bulk of experimental results are located and therefore the bulk of the logic of a paper. This human process of selective focus is mimicked by only finetuning on select parts of the paper. Discussions/Conclusions likely performs better than naively grabbing the middle 50% of the text because the semantic context of a single section is far more likely to be consistent than a smattering of multiple sections. As an analogy, if a person were to be given one chapter of a book as opposed to multiple chapters of a book, it would be far easier for that person to be able to summarize just one chapter of a book. This is because as the size of the text increases, its complexity also increases and therefore the difficulty in synthesizing disparate parts of that text also increases. Therefore, it comes as no surprise that one section which is consistent in tone, voice, and diction is much easier to make sense of then sentences from multiple parts of a paper. This explanation also explains why training on individual sections leads to greater predictive performance than training on the entire text.

3.3.1 Resource Usage

As we can see, as the size of the input increases, the corresponding memory usage on GPU increases dramatically. This made training on a standard 16 GB GPU nearly impossible even on a batch size of 1. Training on select sections decreased the memory footprint since fewer text corresponds to fewer tokens to be stored and therefore a decreased memory footprint. Therefore, training on certain subsets of the paper such as discussions/conclusions section was both more memory efficient and produced better predictions as evidenced by the higher F-score.

Section	Precision	Recall	F-Score	GPU Memory Usage
Intro	0.3581	0.2681	0.2815	24 GB
Middle	0.4621	0.2451	0.3315	28 GB
Disc./Conc.	0.5391	0.2780	0.3616	26 GB
Full Text	0.3821	0.2712	0.2835	33 GB

Table 3. Model Performance for Extractive-Abstractive Networks

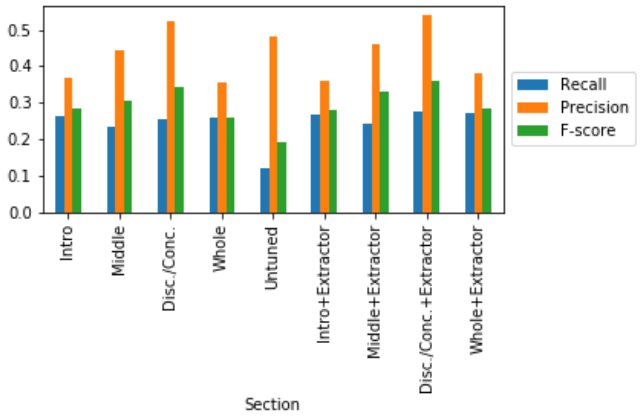

Figure 2. Extractive-Abstractive Model Performance

3.4. On Extractive-Abstractive Hybrid Networks

Based on the results above in section 3.3, we showed that selective processing of the input sequence could yield big performance gains, we then decided to use a more informed method of extracting text to summarize [6]. Instead of simply naively extracting text based on its position in the text, we used a separate transformer network to extract 15 important sentences. The extractive portion of the model effectively points out important sentences to the abstractive portion of the model which then uses those sentences to assemble a summary [6]. This has the effect of simplifying the task of generating a coherent summary from a large input text by reducing the number of candidate sentences. In this way, we reduce the noise of irrelevant contextual information and simply focus the abstractive summarizer (PEGASUS) on only the important details. Figure 2 and Table 3 show the results of the hybrid extractive-abstractive network, and we can see that compared to simply training on the full text (see Table 2.), using extractive summarization on the full text first and then training using PEGASUS yielded a significant increase in precision, recall, and F-score.

Using this result, we then concluded that the same process could be combined with extraction of individual sections. The results are shown in Table 3. We can see that for some sections of the input such as discussions/conclusions and the middle of the text, extractive summarization yields

Section	Precision	Recall	F-Score	GPU Memory Usage
Intro	0.3781	0.3678	0.1760	12 GB
Middle	0.4212	0.1240	0.1894	12 GB
Disc./Conc.	0.4206	0.1375	0.1934	12 GB
Full Text	0.3063	0.1042	0.1528	16 GB

Table 4. Model Performance for Extractive Step on Input + Ground Truth

mild performance gains when compared to just extracting the individual sections. When using extractive summarization on the introduction, there does not seem to be a meaningful performance gain. The intuition for this is relatively easy to grasp. Since introductions are generally shorter than experimental discussions or the main body of the paper, the role of extractive summarization in reducing the number of sentences to consider is minimized. This is much akin to applying a dimensionality reduction technique on an already low dimension problem.

3.4.1 Use of Extractor on Ground-truth

A further modification on this approach is shown in Table 4 where the extractive component is used on both the input as well as the reference text before finetuning. This creates a more memory-efficient training task by essentially creating a proxy task for the abstractive network to learn in the hope that the network would learn enough that the downstream performance on the full task will still be satisfactory. At inference, metrics were computed based on the full abstract. This approach led to a drastically decreased memory footprint as training was able to run on just a 16 GB GPU.

However, performance on the test set for the models trained in this way were lackluster compared to the models trained using full abstracts. The main contributor to these models' lackluster performance is their inability to synthesize text in a way that is coherent and meaningful. Because these models were trained to essentially produce shorter output due to their finetuning task, at inference, they largely produced shorter text than was expected. This is exhibited by these models' extremely poor recall when compared to previous models' results when trained using full abstracts showing that the model is not capturing all of the relevant information present in the text [3].

3.5. Qualitative Evaluation

Overall, summarization in all cases was mostly extractive and there was very little "abstractization" happening. Most summaries simply "copied" large subsequences of contiguous text and rearranged the subsequences in a specific order. Very few texts had unique phrasing or sentence construction that was substantially different from the input text. This is a key limitation of transformer-based methods as they are very good at reproducing patterns but not so proficient in synthesizing understanding and producing new outputs that are different from the input fed into the model. This observation somewhat disproves the hypothesis of the baseline PEGASUS model that doing pretraining with masked important sentences is close enough to the downstream task of producing an abstract summary.

Upon visually inspecting the outputted summaries, we found slightly more unique phrasing and sentence construction when trained on select subsets of text than when training on the full text. However, this is not to say that text summarization was in all cases more abstractive or that we can definitively say that summaries were more extractive in this instance as this was a very modest effect. It is possible that any additional "abstractness" was simply an artifact of random chance. In addition, there did not seem to be a meaningful difference in the "abstractness" of the summaries produced by the extractive-abstractive network as opposed to the baseline models trained.

3.6. Training/Fine-tuning Process

Training/Fine-tuning was performed using HuggingFace utilities for finetuning large-scale transformer language models [10]. The hyperparameters that were tuned were learning rate, batch size, and dropout. Hyperparameter search using the Optuna framework [1] was performed over 100 trials where the function being minimized was the loss evaluated over the validation set and the best model after hyperparameter search was then used on the held-out test set. 1000 training examples were used for finetuning, 128 were used for validation and 500 were used for testing. Training ran for 500 epochs and was stopped early if validation loss did not decrease for 10 epochs. Extractive components, if used, were frozen and the parameters were not finetuned/learned. The loss function used during training is the cross-entropy loss between the generated summary and the ground-truth summary(abstract).

4. Future Investigation

There are several possible extensions for the work presented in this paper. One possbile extension is to finetune the extractive and abstractive component together instead of freezing the extractive layers. This could yield to better performance because those parameters then become learnable on the finetuning task. Another possible extension would be to mix and math different combinations of sections. For instance, combining the introduction and discussion could be an interesting avenue for exploration to see if the model picks up greater context from this input structure. It is noteworthy that PEGASUS model was neither pretrained nor finetuned with human generated abstractive summaries as the labels. Although manually generating such text is labor

intensive, it is likely that model will see improved performance as a result of such endeavor.

5. Conclusions

In conclusion, effectively generating an abstractive summary from a given text is an extremely hard task. Efforts to utilize current Transformer based approaches to this end were predictably computationally expensive. We circumvented that obstacle by utilizing BERT extraction to highlight important sentences as well as pulling sentences from isolated parts of the paper in effort to minimize the RAM requirements to tune the models. We further improved upon both of these strategies by combining the two strategies together. Our efforts revealed that additional fine-tuning and selective indexing of the input resulted in our being able to generate higher ROUGE-L summary level scores than the baseline PEGASUS performance indicating that our efforts were successful. However, we also discovered that fundamental improvements in model architecture and training process would be needed to make generated summaries acquire a truly abstractive flavor.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019. 5

[2] AllenAI. Science parse. https://github.com/UCREL/science_parse_py_api, 2021. 2

[3] GoogleAI. Rouge score. https://github.com/google-research/google-research/tree/master/rouge, 2021. 1, 3, 5

[4] HuggingFace. Pegasus finetuning script. https://gist.github.com/jiahao87/50cec29725824da7ff6dd9314b53c4b3, 2021.

[5] Derek Miller. Leveraging BERT for extractive text summarization on lectures. *CoRR*, abs/1906.04165, 2019. 2

[6] Jonathan Pilault, Raymond Li, Sandeep Subramanian, and Chris Pal. On extractive and abstractive neural document summarization with transformer language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9308–9319, Online, Nov. 2020. Association for Computational Linguistics. 2, 4

[7] PyMUPDF. Pdf parser. https://github.com/pymupdf/PyMuPDF, 2021. 2

[8] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in bertology: What we know about how bert works, 2020. 1, 2

[9] Cornell University. Arxiv dataset. https://www.kaggle.com/Cornell-University/arxiv, 2020. 2

[10] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics. 1, 2, 5

[11] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization, 2020. 1, 2, 3

Student Name	Contributed Aspects	Details
Scott Goldstein	Implementation, Analysis and Report writing	Composed significant portions of the final write up as well as general paper editing. Also supported coding and experiments as needed
Joseph Hirschi	Implementation, Analysis and Report writing	Ran multiple experiments to fine-tune the model with different batches of data from various sections of the academic papers, as well as with extracted sentences. Analyzed results, generated graphs, and contributed to the paper write up.
Aditya Duddikunta	Implementation, Data Preparation, Analysis, Report Writing	Created data preprocessing scripts using PDF parser as well as section parser to clean the data for use in training. Performed several experiments on different slices of data, participated/lead discussions, analyzed results of experiments, and took part in discussions as well as paper writing
Rajneesh Kumar	Implementation, Analysis and Report writing	Ran baseline model and finetuned on multiple batches of extracted text. Implemented finetuning code, results plotting code, analyzed results and took part in discussions and report writing.

Table 5. Contributions of team members.