Project report
Text Summarizer

Timothée Ly, Troy Fau
Group 11

Abstract

The main focus of the report is the implementation of a summarization system based on a pre-trained transformer model. The report details state of the art approaches to text summarization, namely abstractive and extractive summarization and how to use transformers to implement the former. The fine-tuned model is then evaluated using ROUGE metrics. A thorough discussion about the dataset specifications and hyper-parameters choices provides a meaningful insight into the results.

1 Introduction

The aim of this project was to develop an abstractive summarization network. We distinguish two approaches to summarization: extractive and abstractive. Extractive summarization selects the key sentences of a text, which can be done by ranking the sentences of a text in terms of relevance (e.g. by looking for the terms with the highest term frequency). With abstractive summarization, the model extracts the key points from the text, but creates new sentences to summarize the information. The resulting summary can contain words that do not appear in the original text. This method requires a generative model. Naturally, abstractive summarization is harder to implement and more computationally demanding but also yields better results than the extractive approach [1].

We used a pre-trained transformer model and fine-tuned its weights to make it efficient for a summarization task. Pre-training a large language model on a large corpus of data (typically obtained from the web) is a form of unsupervised learning that allows the model to develop an overall knowledge of written text that can then be leveraged for more specialized (so-called downstream) tasks [2]. The chosen model was the smallest GPT-2 [3] model, with 117 million parameters. The fine-tuning was carried out using the Amazon Fine Food Reviews dataset, and ran on a Google Cloud virtual machine equipped with an A100 GPU.

The code was written in `Python` and the main libraries used were :

- `transformers` from **huggingface**
- `pytorch`
- `evaluate`

2 Previous Work

Two methods were considered at first, either train a recurrent neural network (RNN) from scratch [4] or fine-tune the weights of a pre-trained transformer model. We chose the latter as transformer models perform better since, instead of treating the input sequentially (word after word), they treat it as a whole. This non-sequential processing, combined with positional embeddings that encode a token's position in a sequence, make them better at learning long-term dependencies, since they do not need to retain information through a sequence of hidden states, as in the case of RNNs.

The core innovation of the transformer is the self-attention mechanism [5]. RNNs partly alleviated the information loss problem by introducing attention, through which the decoder uses information from all encoder hidden states, instead of only the last one. Information still progresses sequentially through the network. Self-attention operates within the same unit (encoder or decoder), and dispenses with recurrence entirely, allowing each sequence input (i.e. word) to obtain information from every other input, thus directly encoding relations between words.

Treating the sequence as a whole also allows transformers to parallelize processing of a sample during training.

[6] explored using pre-trained models for sequence to sequence tasks (including text summarization), one of which was GPT-2. Pre-trained transformers were also used in [7] as an encoder for an encoder-decoder architecture.

3 Method

3.1 What transformer to choose?

The original transformer architecture [5] is an encoder-decoder. The decoder is similar to the encoder except it masks future inputs and thus functions autoregressively (predicts the next token based on previous tokens), and includes a traditional encoder-to-decoder attention layer after the self-attention layer.

The BERT [8] model is encoder-only, and must be combined with a decoder unit for a task like text summarization. This was done in [7], but would obviously have involved more work for us, as we would have had to choose and tweak the decoder.

[6] used different encoder-decoder setups, and GPT-2. GPT-2 is a decoder-only model, and thus has less parameters overall (117 million) and a much smaller memory footprint. An alternative tried in [6] was sharing parameters between the encoder and the decoder, to reduce parameter count. This would still involve having to manually implement the attention layer within the decoder unit, with non-pre-trained weights. In short, using GPT-2 was the simplest approach for us, though its results in [6] were among the weaker setups.

An example of a dedicated encoder-decoder model is BART [9], but it has more parameters (140 million) than GPT-2. Therefore we settled on using GPT-2. Figure 1 illustrates the model's basic architecture.

3.2 Fine-tuning GPT-2

Fine-tuning a pre-trained model is a form of transfer learning. This method allows end users to save time and money by relying on a model that has already been trained on a large corpus. Most of the model weights are frozen, and only a subset are trained further

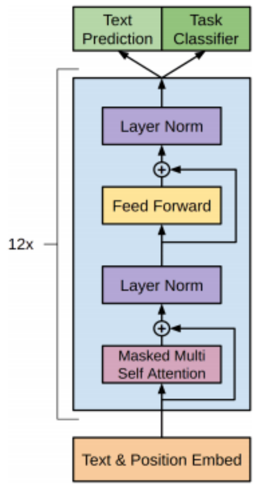

Figure 1: GPT-2 architecture. The transformer block is repeated twelve times. Layer norm is a form of normalization that normalizes across all features of an instance, rather than across all instances of a batch (batch norm).

on data specific to an end user's task. The last layer of GPT-2 is a linear layer, called a Language Modeling Head, which outputs logits that are then *softmax*-ed to predict the next word. The weights we trained are those of this final layer.

To make GPT-2 train for summarization, an input vector must concatenate[10] the review, the string "TL;DR:" and the summary, all in tokenized form. To reduce training time to manageable levels, we set a seed and randomly selected 50,000 samples from the dataset, having already pruned the data of concatenated vectors longer than 512 tokens (Note that the GPT-2 tokenizer uses byte-pair encoding so tokens are not specifically words). These samples were further subdivided into 45,000 training, 2,500 validation and 2,500 test samples. We used a learning rate of 5e-5, based on typical values used by GPT-2 fine-tuning projects, and AdamW as the optimizer. The validation set was used for early stopping, upon detecting an increase in the validation loss. This occurred after 7 epochs of training.

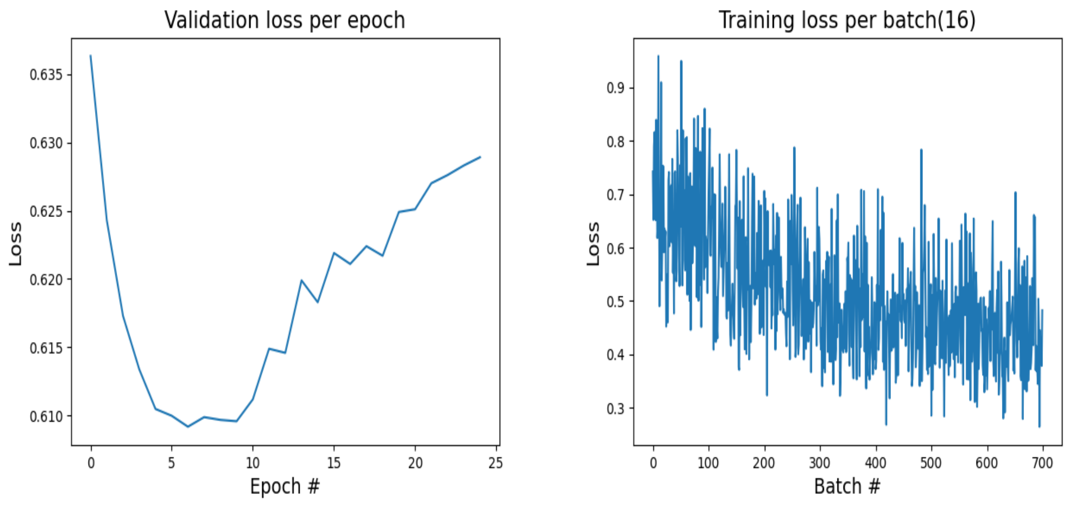

Figure 2: Early stopping with validation loss monitoring.

Once the model was fine-tuned, we then generated summaries from the test set of summaries.

3.3 Summary Generation

It is of paramount importance to discuss the content of the dataset. We computed some statistics on it to better match the summary generation.

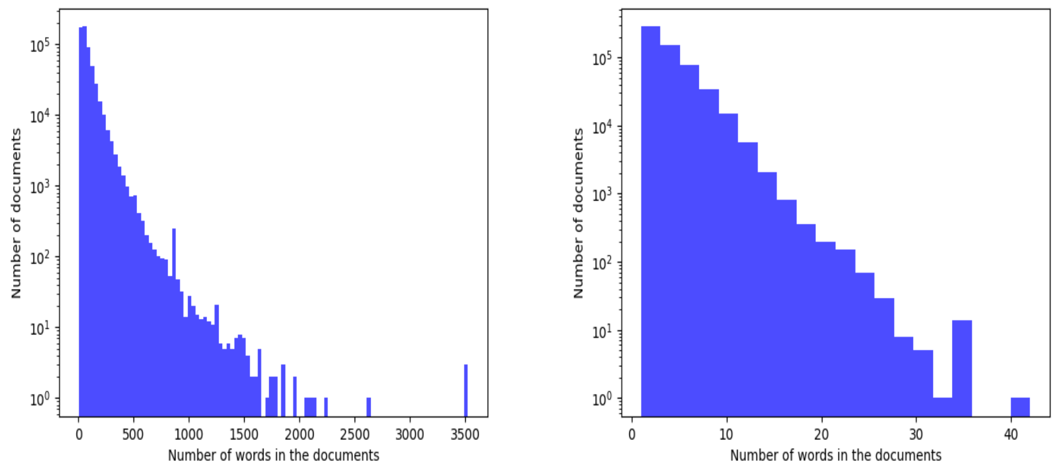

Figure 3: Word distribution of reviews (left) and summaries (right).

	Review	Summary
Average number of words	82.00	4.13
Standard deviation number of words	80.81	2.61
Min number of words	3	1
Max number of words	3526	42
Average number of '!'	0.76	0.34

This step was important to understand the behaviour of our fine-tuned transformer, notably its overuse of the '!' character. An example of a summary generated by our model (without further operation than fine-tuning) for a negative review would be : "Bad !!!!!!".

This gave us some insight about which parameters to set for GPT-2's `generate` function. We compelled the model to generate summaries ranging from 1 to 8 words. To prevent the model from using too many exclamation marks, we forbade the model from re-using any 2-gram in the same summary. This would be detrimental to relevance when generating a longer summary, but the dataset summaries, and those we aim to produce, are very short, hence making it a non-issue.

We then carried out a grid search on the validation set, using 16 different decoding strategy settings (table 1) to select generated probabilities, with 5 different temperature values for each, for a total of 80 combinations. The decoding strategies are detailed in [11].

Decoding method	Values
Greedy search	-
Beam search	2, 4, 6 beams
Sampling	-
Top-K sampling	Top 25, 50, 100
Nucleus sampling	Top 0.6, 0.7, 0.8, 0.9
Top-K, nucleus	Top 50, 0.6; 50, 0.7; 50, 0.8; 50, 0.9

Table 1: Hyperparameter search. For each decoding method and value, we evaluated with 5 different temperatures: 0.2, 0.4, 0.6, 0.8, 1. A lower temperature skews the distribution produced by *softmax* towards higher probability tokens.

3.4 Evaluation

Evaluating the quality of the recommendations given by the system can be difficult since it can be seen as subjective. The wording or even the information left out are very dependent on the person in charge of summarizing a text. Hence, summarization accuracy - *i.e.* the number of summaries exactly similar to the reference ones - could give a very negative results while the summarization engine performs well. Obtaining a summarization accuracy close to 1 is impossible (unless the dataset is trivial and the model overfit). Therefore, we have to find another way to assess our method.

ROUGE ROUGE stands for *Recall-Oriented Understudy for Gisting Evaluation* [12]. It is a set of metrics especially used for language processing tasks. We will detail two of them:

- ROUGE-n : The overlap of n-grams between the summary and the reference summary. In that case :
$$\text{precision-}n(s,s_{ref}) = \frac{\text{Number of common n-grams}}{\text{Number of n-grams in the summary}}$$
$$\text{recall-}n(s,s_{ref}) = \frac{\text{Number of common n-grams}}{\text{Number of n-grams in the reference summary}}$$

  For ROUGE-n, recall can be understood as whether or not the summary contains most of the n-grams of the reference summary while precision ensures that the summary is not noised with many irrelevant words.

- ROUGE-L : The Longest Common Subsequence (LCS) between the summary and the reference summary. This metric is more complex in its definition. A subsequence of a sentence $(a_0, \dots, a_n)$ is a sequence $(a_{i_0}, \dots, a_{i_k})$ where $0 \leq i_0 < \cdots < i_k \leq n$. For example, take the summary sentence "The cat sleeps under the bed" and "The cat was sleeping under the blue bed.". The LCS is "The cat under the bed". With this definition, we can extract the LCS. To expand this definition to a summary of multiple sentences, each sentence in the reference summary is paired with the best sentence candidate in the summary.

$$\text{precision-}L(s,s_{ref}) = \frac{\sum_{i=1}^{u} \text{LCS}(r_i, C)}{\text{Number of n-grams in the summary}}$$
$$\text{recall-}L(s,s_{ref}) = \frac{\sum_{i=1}^{u} \text{LCS}(r_i, C)}{\text{Number of words in the reference summary}}$$

where u is the number of sentences in the reference summary, r_i a reference sentence and C the set of all sentences from the summary.

ROUGE, introduced in 2004, is still widely used to evaluate the results of language processing networks. Compared to a basic accuracy computation, these metrics are on a much higher semantic level. Intuitively, they correspond to the first step when someone is asked to compare how close two sentences are, one starts by comparing the co-occurring words.

We used six ROUGE metrics to measure the performance of our fine-tuned network: precision and recall for ROUGE-1, ROUGE-2 and ROUGE-L. Our best hyperparameter combination according to the summed values was with beam sampling using 6 beams and any temperature. Second best was top-100 with a temperature of 0.4. Table 2 shows the results of our evaluation on the test set with these settings.

Metric	R1 prec	R1 rec	R2 prec	R2 rec	RL prec	RL rec
Beam search	0.128	0.1707	0.041	0.0548	0.118	0.1597
Top-100 sampling	0.1338	0.1623	0.0285	0.0351	0.1309	0.1576

Table 2: ROUGE performance of our best models on the test set. The beam search uses 6 beams; the top-100 sampling uses a temperature of 0.4.

4 Discussion and Conclusion

Results The results were disappointing when assessed with ROUGE metrics. Even though we used grid search to find the best settings, it was an unrealistic expectation to find expect many common words between summaries that are around 4 words long. ROUGE-2 (2-gram overlap) in particular performed terribly compared to ROUGE-1.

Nonetheless, when we manually assess the relevance of the summaries produced by our model, they are of greater quality than what ROUGE suggested.

Furthermore, thanks to early stopping the model did not overfit, as we can observe on defective reference summaries (though our own summary is also subpar and contains some irrelevant content) :
Text : "It doesn't take much to make a wonderfully fragrant cup of tea – exceptional :-)" | Reference Summary : "happy holidays!" | Our Summary : "Yummy tea, great price!!"

It is worth drawing attention to the existence of these subpar reference summaries, which pollute the dataset. When they appear in the test set, they degrade both ROUGE scores and assessed relevance. Combing through the data to remove poor summaries was of course a task that would have been far too time-consuming.

An example of a mediocre score but high relevance: Text: "This ginger is consistent quality with a good taste. Very tender little cubes of candied ginger. Great for tea or just snacking." | "Reference Summary: "Just good taste" | Our Summary: "Good quality ginger, great taste!!"

The reference summary is arguably too succinct, and does not even mention the product. Our summary mentions the ginger and its quality. ROUGE-1 recall is 0.6667 and precision 0.4; ROUGE-2 score is 0 for both, while ROUGE-L gives a recall of 0.6667 and a precision of 0.4. Mediocre values, especially for precision, but that is because our summary adds useful information.

Recall is indeed often higher than precision; this is because our summaries are often longer than the reference summaries, increasing the precision denominator.

Beam search with 6 beams performed slightly better than top-100 sampling according to the ROUGE metrics. Assessed manually, beam search seemed to complete sentences more often, while top-100 sampling had more cases of summaries ending with trailing words. Both strategies produced many exclamation marks.

Limitations The first limitation we encountered was purely practical: even using the GPT-2 model with the smallest number of parameters, it still took an excruciating amount of time to train 1 epoch (758 hours for the full dataset on a CPU). We therefore had to reduce both the number of samples from the dataset and the overall dimension of the input: we trained using samples with a combined review and summary length of 512 tokens, whereas the model can accept vectors of up to 1024 tokens as input. Using a Nvidia A100 GPU, we managed to get training time down to around 30 minutes per epoch, but the usage of this powerful GPU was limited by the amount of free money offered by Google Cloud.

While very relevant for this task, ROUGE does not fix the negation semantic issue. "I absolutely don't recommend this article." and "I absolutely recommend this article." (reference) have a ROUGE-1 recall of 1 and ROUGE-1 precision of 5/6 which indicate a good translation, even though that is clearly not the case. However, this issue is mitigated by the computation of successive ROUGE-n and actually does not happen often. A more serious issue with ROUGE is that it is a better fit for extractive summarization than for abstractive summarization, simply because abstractive summaries can consist of words that do not belong to the original sentence, and synonyms are not accounted for in ROUGE. All in all, it is quite lacking in informativeness about the factual accuracy of a summary. Using a fact-checking network is a new approach [13] to assessing language model capabilities, but the results are still subpar.

Conclusion Fine-tuning a pre-trained transformer for a given task is a very powerful way to get good results without necessitating days of training weights on an enormous amount of data. The choice of transformer model and of the hyperparameters is decisive for model performance. In our case, we observe decent results when manually assessed but overall disappointing ROUGE and parasitic behaviour due to inherent structure of the dataset (small summaries, frequent exclamations points at the end of phrases that strongly biased the fine tuning). Picking another model rather than GPT-2, such as an encoder-decoder architecture, might also have improved the summaries obtained, though it would have involved more parameters and fine-tuning.

5 How to run

All the instructions are included in the `Readme.md` in the repository. The repository is public.

References

[1] Liwei Hou, Po Hu, and Chao Bei. "Abstractive Document Summarization via Neural Model with Joint Attention". In: *Natural Language Processing and Chinese Computing*. Ed. by Xuanjing Huang et al. Cham: Springer International Publishing, 2018, pp. 329–338.

[2] Colin Raffel et al. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In: *J. Mach. Learn. Res.* 21.1 (Jan. 2020). ISSN: 1532-4435.

[3] Alec Radford et al. *Language Models are Unsupervised Multitask Learners*. 2019.

[4] Ramesh Nallapati et al. *Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond*. 2016. arXiv: 1602.06023 [cs.CL].

[5] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[6] Sascha Rothe, Shashi Narayan, and Aliaksei Severyn. “Leveraging Pre-trained Checkpoints for Sequence Generation Tasks”. In: *Transactions of the Association for Computational Linguistics* 8 (2020), pp. 264–280. DOI: 10.1162/tacl_a_00313. URL: https://aclanthology.org/2020.tacl-1.18.

[7] Yang Liu and Mirella Lapata. *Text Summarization with Pretrained Encoders*. 2019. arXiv: 1908.08345 [cs.CL].

[8] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423.

[9] Mike Lewis et al. “BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 7871–7880. DOI: 10.18653/v1/2020.acl-main.703. URL: https://aclanthology.org/2020.acl-main.703.

[10] Urvashi Khandelwal et al. “Sample Efficient Text Summarization Using a Single Pre-Trained Transformer”. In: *CoRR* abs/1905.08836 (2019). arXiv: 1905.08836. URL: http://arxiv.org/abs/1905.08836.

[11] Gian Wiher, Clara Meister, and Ryan Cotterell. “On Decoding Strategies for Neural Text Generators”. In: *Transactions of the Association for Computational Linguistics* 10 (Sept. 2022), pp. 997–1012. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00502. eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00502/2043936/tacl_a_00502.pdf. URL: https://doi.org/10.1162/tacl%5C_a%5C_00502.

[12] Chin-Yew Lin. “ROUGE: A Package for Automatic Evaluation of Summaries”. In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 74–81. URL: https://aclanthology.org/W04-1013.

[13] Yuhui Zhang, Yuhao Zhang, and Christopher D. Manning. *A Close Examination of Factual Correctness Evaluation in Abstractive Summarization*. 2020.