Pre-Training Reformer Language Models for Abstractive Summarisation

Sat 1st Aug, 2020

Samyak S Sarnayak*, Varun P†, Pranav Kesavarapu‡ and Dr. Jayashree R§
dept. Computer Science and Engineering, PES University
Bengaluru, India
Email: *samyakssarnayak@pesu.pes.edu, †varunp@pesu.pes.edu, ‡pranavkesavarapu@gmail.com, §jayashree@pes.edu

***Abstract*—Abstractive summarisation for long sequences has been a challenging area for research. The current most commonly used models revolve around the transformer architecture. These do not perform well on long sequences and generally run into memory issues most frequently amongst other problems. The Reformer architecture addresses some of the issues of transformers. This work discusses the issues regarding the transformer architecture, how the reformer, along with other techniques, attempts to solve these issues and then downstreams the Reformer Language Model for the task of abstractive summarisation and evaluates the performance of the pre-trained and non pre-trained reformer language model against the BigPatent dataset. For pre-training, a self-supervised approach of generating the summary from documents based on the ROUGE-1 F1 score is used, which is named Gap Sentences Generation (GSG). We also investigate the performance variation of GSG with varying input size and Reformer with varying hyper-parameters.**

***Index Terms*—Abstractive Summarisation, Reformer, Transformer, Self-supervised objective, Language modelling, Gap Sentences Generation, Pre-Training**

I. Introduction

In NLP, summarisation is of two kinds, *extractive summarisation* and *abstractive summarisation*. In *extractive summarisation*, the model aims to identify and extract important subsets of sentences that explain the context of the actual document concisely. The summary produced would consist of parts of sentences present in the actual document. In *abstractive summarisation*, the model tries to understand the document and explain it in its own words. The summary produced would be a concise explanation of the document in mostly unique phrases. Abstractive summarisation tends to be more coherent and similar to that of a human summarisation. The drawback of abstractive summarisation is that it tends to be relatively computationally intensive. For that reason, extractive summarisation tends to be faster than abstractive summarisation and abstractive summaries tend to be much more intuitive to read. In this paper, we perform abstractive summarisation using the Reformer architecure with and without pre-training. Most of the recent work deals with use of transformers and other models to generate coherent abstractive summaries.

A. The Tranformer Model

Transformers [1] have shown immense success in natural language understanding and generation tasks. Transformer based models are state of the art or competitive with them for machine translation, question answering and language modelling. Large transformer based pre-trained models are commonly used as a base model for fine-tuning on specific tasks since they show good generalisation ability. Transformers use an attention mechanism to attend to the entire sequence at once. The attention mechanism used in transformers consists of first generating a set of keys K, a set of queries Q and values V corresponding to the keys. The output is determined by the values which are selected from the keys which are closest to the queries. This requires computation of the dot-product between queries and keys QK^T which is of size $(length, length)$ where length is the input sequence length. This leads to a quadratic memory and time complexity in terms of the input length. Thus, as the input size is increased the computations can become large very quickly which means that the input size has to be limited to train models in a reasonable amount of time. Larger sequences have to be used to get better results, but training transformers on large input sizes can only be done by highly equipped corporations and laboratories. Some large pre-trained transformers models have such a large memory requirement that they cannot fit into a single accelerator, such as a GPU, for fine-tuning. The high memory requirement is further increased by the feedforward sub-layer which can have thousands of units. Also, during training, the activations or outputs from every layer has to be cached so that it can be used in the backpropagation step which contributes to a significant amount of the memory usage; this increases with the number of layers. Therefore, the main limitations of transformer is its performance on long sequences of data.

B. The Reformer Model

The Reformer architecture [2] deals with the limitations of Transformer by using several techniques to reduce the time and memory complexity of the model. Specifically, reformer uses three major techniques, namely Locality Sensitive Hashing (LSH) based Attention, reversible layers and feed-forward chunking. **Reversible layers** were introduced by Gomez et al. [3], the basic idea is that activations (outputs) for a layer can be calculated when needed during backpropagation using the activations of the next layer. This is achieved by using a pair of inputs and outputs for every layer which, in case of transformers, is the attention and the feed-forward layers. Thus, this reduces the amount of memory required by n_l which is the number of layers. While reversible layers reduce the memory required by multiple layers, individual layers in the model can be large, especially the feed-forward layers which can have thousands of units. To reduce memory required by feed-forward layers, the computation in them can be split into chunks and each chunk can be processed individually. This is possible due to the fact that computations in the feed-forward layers are not time/sequence dependent. By processing the chunks sequentially, memory requirements can be reduced by a significant amount. This is known as **feed-forward chunking**. Transformers generally use dot-product attention which involves finding the values (V) using keys (K) which are closest to some given queries (Q), all of which are vectors of shape $(batch\_size, sequence\_length, d_{model})$. Q, K and V all come from the same given sequence, the input sequence, through three different sub-layers. In LSH attention, queries and keys are shared i.e., K = Q requiring only two layers. Kitaev et al. [2] showed that sharing QK needs to no loss in performance. **Locality-senstive hashing (LSH)** is a hashing scheme for vectors in which a random rotation is applied to each vector and based on the rotated position the vectors are assigned to different buckets/chunks, the corresponding bucket is then considered as the hash value. Performing the random rotations a few times in parallel, usually 2-8, leads to a distribution in which vectors that lie close to each other are present in the same chunk i.e., they have the same hash. In LSH Attention, the queries, or keys, are first hashed using LSH to get the bucket number for each query. The queries are then sorted by the bucket number and inside each bucket, the queries are sorted by their position in the sequence. After bucketing, a set of m queries in a bucket are allowed to attend to each other and the previous bucket. According to Kitaev et al. [2], m is set to $m = \frac{2l}{num\_buckets}$. LSH Attention reduces the time and memory complexity of attention from quadratic to linear. Thus, Reformer helps in dealing with limitations of transformer by being efficient for longer sequences.

In standard transformer models, the positions for sequences have to be injected using position embeddings. The position embeddings consist of vectors of length d_{model} for each and every token in sequence i.e., for all positions 1, 2..., *length* which is stored in a position embeddings matrix X. On long sequences, this position matrix could be very large in the order of hundreds of millions of parameters. To reduce the number of parameters further, we use **Axial Position Embeddings** in which the embedding matrix is factorized into two matrices, X^1 of size (d^1, n^1) and X^2 of size (d^2, n^2). The embedding matrices are indexed using the following scheme.

$$X_{i,j} = \begin{cases} X^1_(i,k) & i < d^1 \ with\ k = j\ mod\ n^1 \\ X^2_(i - d^1, l) & i \geq d^1 \ with\ l = \lfloor \frac{j}{n^1} \rfloor \end{cases}$$

Axial position embeddings dimensions (d^1, d^2) are set such that $d_{model} = d^1 + d^2$ and the position shape (n^1, n^2) is set such that $length = n^1 \times n^2$. The number of parameters is reduced significantly. For example, a model with $d_{model} = 2^9 = 512$ and $length = 2^16 = 65536$ will require $512 \times 65536 = 33554432 = 33M$ parameters with the standard position embeddings. Using axial positon embeddings with parameters $d^1 = 256, d^2 = 256, n^1 = 2^8, n^2 = 2^8$ will require $256 \times 256 + 256 \times 256 = 131072 = 0.13M$ which is a very significant reduction. In this paper, we combine the Reformer architecture with axial position embeddings.

C. Pre-Training

Pre-training deep neural networks is a popular technique used to create a general model that works well on many downstream tasks. For example, a pre-trained *BERT* [4] language model can be used in language translation, question answering and text summarisation all by fine-tuning the same pre-trained BERT model in a relatively short amount of time. Fine-tuning is the process of training a pre-trained model further on a specific downstream task. Since pre-training is done on a large amount of data, they mainly use a *self-supervised* method of training. Next token prediction, where the model learns to generate the next token given the text upto that token, and *mask language modelling* (MLM), where some words in the text are removed or replaced and the models learns to restore the masked tokens, are two commonly used self-supervised techniques used in pre-training language models. Pre-trained language models have shown a high degree of success in natural language processing and generation tasks. The main advantage of pre-training is the inherent generalisability present in them which leads to sample efficiency i.e., high accuracy can be obtained on a specific task by using a (relatively) small number of training samples. Khandelwal et al. [5] showed that pre-training a decoder-only transformer language model leads to high sample efficiency for abstractive summarisation. It was also shown that summarisation, which is a sequence-to-sequence task, can be modelled using a decoder-only network with the appropriate inputs.

D. Gap Sentence Generation

Self-supervised objectives generally used in pre-training language models such as next token prediction and MLM mask individual tokens or words, which is very different from summarisation where entire sentences have to be generated using the context of the document. Zhang et al. [6] introduced PEGASUS, Pre-training with Extracted Gap-sentences for Ab-stractive summarisation Sequence-to-sequence models, where a self-surpervised objective named Gap Sentence Generation (GSG) is used. In GSG, entire sentences are masked instead of individual tokens. This resembled the task of summarisation more than masking individual tokens. Our implementation of GSG is described in section III-E.

II. Literature Review

In abstractive summarisation, using LSTMs, RNNs along with sequence-to-sequence models [7] was a common technique in producing state-of-the-art results. Encoder-decoder architecture built up on these to produce much better results. Recurrent models have a drawback of being computationally expensive. To improve these, the attention mechanism [8] was introduced to increase the performance; but as it depends on recurrent models, they are still computationally intensive. In 2017, Vaswani et al. [1] brought about a new revolutionary architecture called Transformer that did not depend on recurrence; because of this, the performance of these transformers were much higher than its predecessors with much lesser training time. This led to the rise of extremely potent models such as BERTSUMABS [9] and T5 [10]. Transformers LMs are popular for summarisation tasks [11]. Some state-of-the-art summarisation models that do not use transformers are Pointer-Generator networks [12] and reinforcement learning based abstractive summarisation model [13].

BERTSumAbs This [9] is an abstractive summarisation model that used a pre-trained Bidirectional Encoder Representations from Transformer (BERT) model [4] which was modified for summarisation. BERTSumAbs uses multiple `[CLS]` tokens which help it understand the semantics and structure of sentences better. It also uses different optmisers and learning schedule for the encoder and the decoder and they are trained in a two-stage fashion.

T5 Transformer Text-To-Text Transfer Transformer [10] is a transformer which was built for transfer learning. The T5 transformer, an encoder-decoder architecture, was trained on a extremely large amount of data called C4 (Colossal Cleaned Crawled Corpus), to test the limits of transfer learning in NLP. The encoder-decoder architecture was designed to be similar in dimensions to BERT. In T5, the inputs and outputs are both texts i.e., there is no additonal information given to the model.

Extractive and Abstractive Neural Document summarisation with Transformer Language Models Transformers have been used in many successful models in NLP. A transformer encoder-decoder architecture was used to summarize text of separate parts of several long documents. The document would be split into parts such as *introduction, extracted sentences , and/or other extracts* [11]. The transformer can be used both as an extractive summariser and an abstractive summariser.

Pointer-Generator By addressing the drawbacks of sequence-to-sequence such as inability to reproduce facts correctly and their tendency of repetition, the Pointer-Generator [12] introduces a novel hybrid pointer-generator. Here, the *pointer* copies essential words which are essential to the context, while the *generator* produces new sentences using the captured words. A *coverage* is also used which minimizes repetition by keeping track of sentences which have already been covered.

RL Abstractive summarisation Supervised learning models often exhibit *exposure bias*, i.e. the bias arising from the model being exposed only to the training data distribution instead of its own prediction. One of the best ways to tackle this is to use reinforcement learning. This model [13] uses a novel architecture called intra-attention coupled with token generation and pointe , which prevents repetition, and a training method that uses reinforcement learning to generate new sequences.

III. Methodology

A. Reformer LM

We use a Reformer Language model (ReformerLM) as the primary model in this paper. ReformerLM is a decoder-only model, this is due to the fact that LSH requires queries (Q) and keys (K) to be the same - in an encoder-decoder model Q and K come from different sequences. This LM uses all the features of reformer including LSH, reversible layers and FF chunking. Axial position embeddings is also used. Note that not all layers of the model use LSH, alternating LSH and chunked dot product attention were used to keep the benefits of both.

B. Summarisation using Reformer LM

Since it is a language model, it can only perform language modelling tasks such as next word/sentence prediction. Language models are trained auto-regressively, the sentence upto the last predicted word is given as input and the model predicts the next word. Similarly, during evaluation/testing, a starting sentence is given and the model predicts the next sentence(s) by predicting one word at a time after which the generated sentence along with the predicted word is fed back as input to the model. This is the greedy search method of language modelling. While this technique can lead to the model learning the language representation effectively, it is not suitable for summarisation which is a sequence-to-sequence generation task. The model needs to learn to convert an input of a large size to a much smaller sized output. Thus, summarisation needs to be modelled as a language modelling task. The way this is done is by appending the summary to the text which is then given as the input. The text and summary are separated by a special token which we will denote with `<SUMM>`. The modification here is that loss is computed only over the summary. The inputs are padded, with zeros, to a fixed length since the model takes a fixed length input. The input along with the loss mask is as given in Fig. 1.

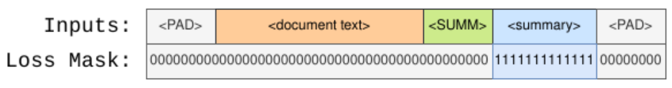


Fig. 1. Inputs and loss mask for ReformerLM summarisation

The padding scheme used was random padding on both sides of the inputs, this was done to prevent the model from learning positions of the text instead of the content.

C. Reformer Encoder-Decoder summarisation

A Reformer encoder-decoder sequence-to-sequence model uses reversible layers but not LSH, instead a simple dot product attention is used for the reasons described in section III-A. The summarisation task for this reformer is a simple sequence-to-sequence task where the input is the document and target output is the summary with a diagonal loss mask for attention i.e., it is allowed to attended to the previous tokens but not tokens in the future.

D. Pre-training

Khandelwal et al. [5] showed that pre-training a Transformer Language Model (TransformerLM), which is a decoder-only transformer network, leads to a model which can be fine-tuned on a downstream summarisation task even with a limited number of samples. Since reformer is a transformer and is best used as a decoder-only model, in this paper we pre-train the reformer model before fine-tuning it to a summarisation task.

E. Gap Sentence Generation

Zhang et al. [6] introduced a self-supervised pre-training objective, named *Gap Sentence Generation* (GSG), specifically designed to be suitable for downstream summarisation tasks. In GSG, a pseudo-summary of the input document is generated by extracting the top sentences i.e., sentences which are most important to the document. This resembles the downstream task of summarisation more closely compared to next word prediction or masked language modelling, in which individual tokens are replaced. The number of sentences to be extracted from the document is defined by a parameter called *Gap Sentence Ratio* (GSR) which is the fraction of number of sentences to be selected from the document. As in [6], importance of a sentence is approximated using *ROUGE-1 F1* score which is calculated between a sentence s and the rest of the document $S - s$. Algorithm-3 shows the algorithm for obtaining the top m sentences from a set of sentences, the score is greedily maximised between the set of selected sentences S and the rest of the document $sentences - S$, thus giving a set of sentences which together represent the document in the best way. Algorithms 1 and 2 define helper functions for 3 which have been optimised using memoisation.

F. Datasets

1) Wiki: The Wiki corpus dataset [14] we used is a collection of all English articles from Wikimedia. The wikimedia dump consists of over 3.4 million articles, totalling over 17 Gigabytes in size [1]. Due to resource constraints, we trained our models on a small subset of the corpus.

[1]The Wikimedia dumps can be found here: https://dumps.wikimedia.org/enwiki/latest/

Algorithm 1: Memoization-based Rouge score for single hypothesis and multiple references

```
input  : A set of reference sentences $R$, a
         hypothesis sentence $h$ and a mapping of
         $(hypothesis, reference) \to rouge_{F1}$
         denoted by $T$
output: Aggregate Rouge score for all references
$S = \varnothing$
for $r \in R$ do
    if $(h, r) \in T$ then
        $s = T[h, r]$
    else
        $s = rouge(h, r)$
        $T[h, r] = s$
    end
    $S = S \cup s$
end
return $aggregate(s)$
```

Algorithm 2: Memoization-based Rouge score for multiple hypotheses and multiple references

```
input  : A set of reference sentences $R$, a set of
         hypothesis sentences $H$ and a mapping of
         $(hypothesis, reference) \to rouge_{F1}$
         denoted by $T$
output: Aggregate Rouge score for all hypotheses
        and references
$S = \varnothing$
for $h \in H$ do
    $s = rouge\_multiple\_references(h, R, T)$
    $S = S \cup s$
end
return $aggregate(S)$
```

2) BigPatent: BigPatent summarisation dataset was introduced by [15]. This paper chooses to evaluate the reformer model against the BigPatent dataset because most of the standard de-facto datasets for abstractive text summarisation like CNN/Daily Mail dataset [13] belong to the news domain i.e, it is a compilation based on news articles. The problem with these datasets is that it being a compilation based on news articles the contents required for the summary generally appear in the starting and ending of the texts apart from this there are numerous reasons few of which are mentioned in [16]. The news article based datasets do not necessarily evaluate the models understanding of the text especially on longer sequences. The BigPatent dataset [15], a collection of patent documents for filed and given patents in the United States of America with their respective human written abstractive and coherent summaries. Compared to other summarisation datasets this paper chooses to evaluate the Reformer LM for the task of abstractive summarisation on the BigPatent dataset [15] because of the following reasons.

1) Summaries are longer than the commonly used

Algorithm 3: Sequential Gap Sentence Generation

```
input : The set of all sentences $sentences$ from
        the document and the number of top
        sentences to extract $m$
output: A set of sentences $S$ which are the top $m$
        sentences
$S = \varnothing$
$table = Dictionary((hypothesis, reference) \rightarrow rouge_{F1})$
for $j$ in 1 to $m$ do
    // $S_j$ will store ROUGE scores
    $S_j = \varnothing$
    for $s_i \in sentences$ do
        if $s_i \notin S$ then
            $tmp = S \cup s_i$
            $S_j = S_j \cup rouge\_memoize(tmp,\ S - tmp,\ table)$
        else
            $S_j = S_j \cup \{-1\}$
        end
    end
    $k = argmax(S_j)$
    $S = S \cup sentences_k$
end
```

datasets.

2) In the input document the important features are evenly distributed throughout the document.
3) The texts in the summaries have fewer, lesser and shorter extracted parts from the document.

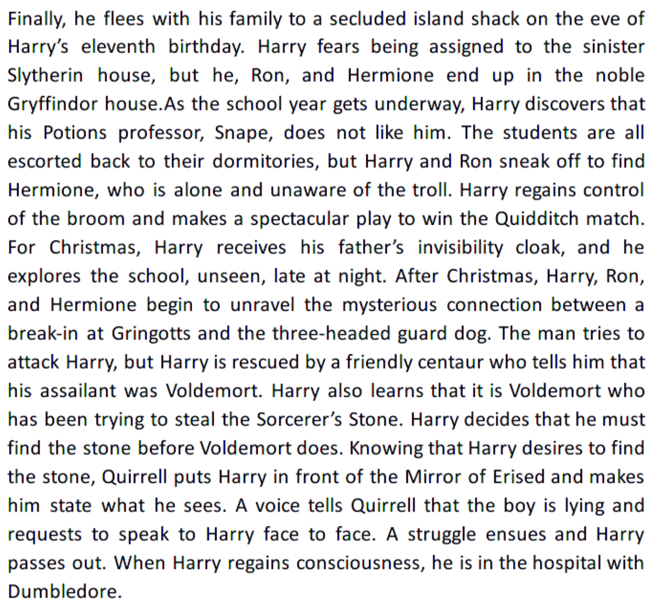
G. Experiments

In this paper, we experiment with pre-training a reformer LM and then fine-tuning it. It is to be noted that there was resource constraint consistently which consequently lead to the model being trained on a subset of the datasets and for a shorter period of time. First, we pre-train a Reformer LM on the English Wikipedia corpus using GSG i.e., sentences are masked from the input and given as the summary. As show in section III-B, the inputs consists of the summary appended to the document with a special token as delimiter. The pre-trained model is then finetuned for summarisation on the BigPatent dataset. For both pre-training and finetuning, a maximum sequence length of 24576 (24K) tokens was used. All models were trained with the help of the ADAM optimiser [17] with a learning rate of 1×10^{-4}. The Reformer LM used the following hyperparameters: $d_{model} = 512, n_{layers} = 6, n_{heads} = 8, n_{hashes} = 2$. For the axial position embeddings, $n^1 = 96, n^2 = 256$ was used. The entire model consisted of $3.3M$ parameters.

Pre-trained and non-pre-trained models were compared on the BigPatent summarisation task. Outputs were generated using *greedy search decoding* which works as follows. The input given is used to generate the next token, which is then appended to the input and fed back into the model. The model then generates the next token, this is continued till a maximum number of tokens or when the end sequence token is predicted which, in our case, is the padding token. The input to our model when predicting consists of the document text with the summary token `<SUMM>` appended to the end; this is done to prompt the model to generate a summary.

Trained models are evaluated qualitatively by looking at the generated summary and comparing it to the text. Quantitatively, generated summaries are evaluated using *BiLingual Evaluation Understudy (BLEU)* [18] scores - BLEU-1, BLEU-2, BLEU-3 and BLEU-4; and *Recall-Oriented Understudy for Gisting Evaluation (ROUGE)* [19] scores - ROUGE-1 (which was also used in GSG), ROUGE-2, ROUGE-3, ROUGE-4, ROUGE-L and ROUGE-W.

Additionally, various experiments are conducted with GSG to quantitatively and qualitatively evaluate the pseudo-summaries generated by it and its time complexity.



Finally, he flees with his family to a secluded island shack on the eve of Harry's eleventh birthday. Harry fears being assigned to the sinister Slytherin house, but he, Ron, and Hermione end up in the noble Gryffindor house.As the school year gets underway, Harry discovers that his Potions professor, Snape, does not like him. The students are all escorted back to their dormitories, but Harry and Ron sneak off to find Hermione, who is alone and unaware of the troll. Harry regains control of the broom and makes a spectacular play to win the Quidditch match. For Christmas, Harry receives his father's invisibility cloak, and he explores the school, unseen, late at night. After Christmas, Harry, Ron, and Hermione begin to unravel the mysterious connection between a break-in at Gringotts and the three-headed guard dog. The man tries to attack Harry, but Harry is rescued by a friendly centaur who tells him that his assailant was Voldemort. Harry also learns that it is Voldemort who has been trying to steal the Sorcerer's Stone. Harry decides that he must find the stone before Voldemort does. Knowing that Harry desires to find the stone, Quirrell puts Harry in front of the Mirror of Erised and makes him state what he sees. A voice tells Quirrell that the boy is lying and requests to speak to Harry face to face. A struggle ensues and Harry passes out. When Harry regains consciousness, he is in the hospital with Dumbledore.

Fig. 2. Summary extracted by GSG from a Harry Potter book.

IV. Results

A. Gap Sentence Generation

GSG was implemented and first tested on a section of a book, Harry Potter. The pseudo-summary show in Fig. 2 was extracted using the sequential algorithm in 3 on the Harry Potter book. It can be seen that the pseudo-summary captures most of the important information from book, with major events summarised in a sentence.

To evaluate the time complexity of algorithms 1, 2 and 3 we use the following procedure. Random sentences with k characters are generated by sampling k characters from a set of alphabets with spaces and full stops where the probability of a space is 5 times that of other characters.

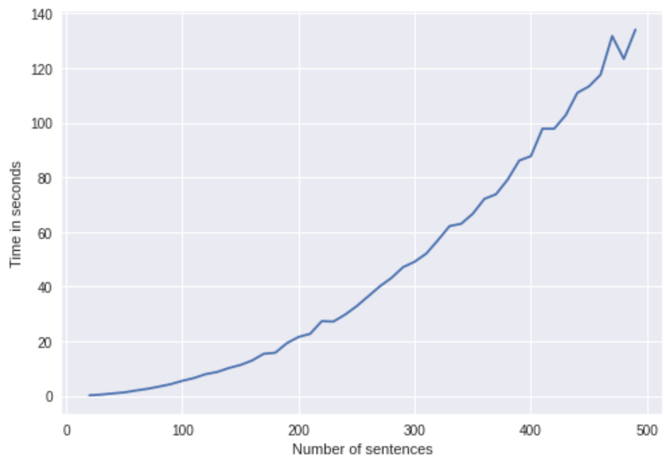


Fig. 3. Time complexity of sequential GSG.

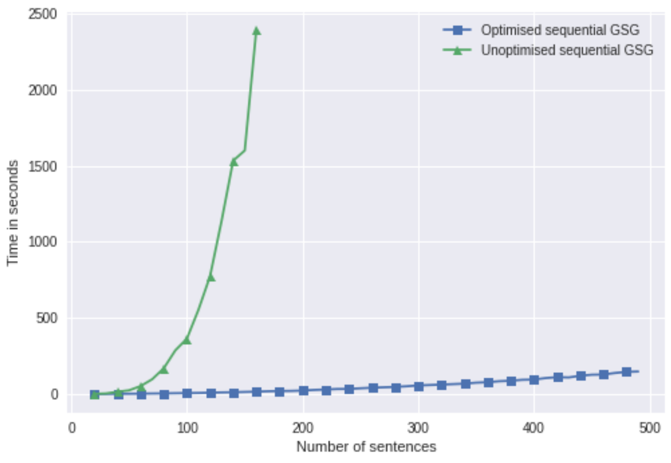


Fig. 4. Time complexity of DP table optimised and un-optimised sequential GSG.

Random sentences with k varying from 2000 to 50000 are generated and GSG is applied to all of them while profiling the time taken to generate summaries. It can be seen in Fig. 4 that the time complexity of sequential GSG is close to quadratic while the time taken by independent GSG in Fig. 5 is lower than that of sequential. Fig. 3 shows that time complexity reduced very significantly after optimisation, from highly exponential to nearly quadratic. Although the independent GSG is faster, sequential GSG generates better summaries and hence it was chosen for further experiments.

B. Performance Evaluation

To study the performance impact of hyper-parameters, we vary the parameters and examine the memory used in training. Memory usage is measured by first resetting the run-time, loading the model and then training the model for one iteration. The memory requirements are also tested by turning off features of Reformer such as LSH Attention and reversible layers, this shows the effect these techniques have on memory.

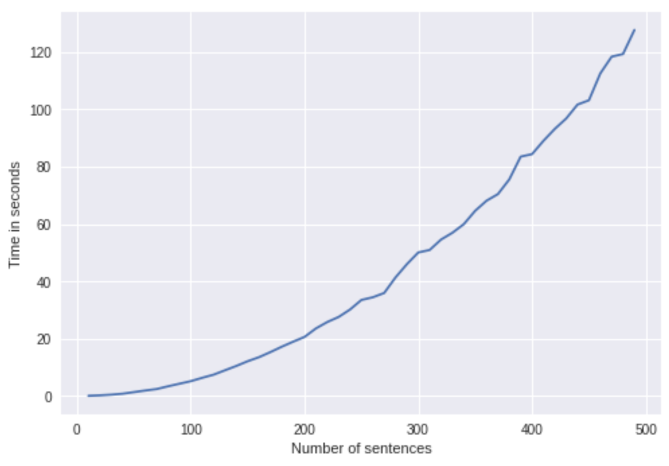


Fig. 5. Time complexity of independent GSG.

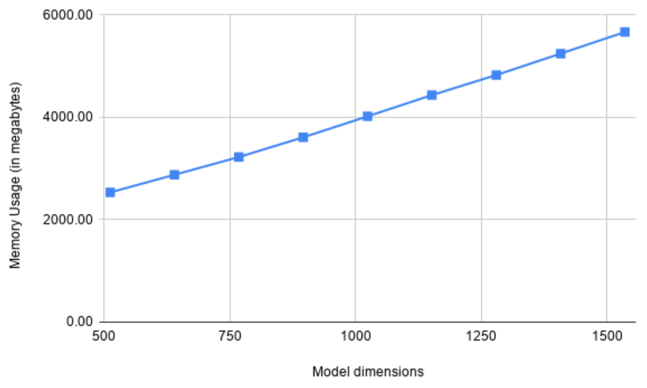


Fig. 6. Memory usage as model dimension is increased. Memory usage increases linearly, but stays under $6GiB$ even for a model with 1538 dimension.

A baseline model with the following hyper-parameters was used in this section (unless stated otherwise):

$$\begin{aligned} d_{model} &= 512 \\ n_{layers} &= 6 \\ n_{heads} &= 8 \\ n_{hashes} &= 2 \\ depth &= 6 \\ batch\_size &= 1 \\ bucket\_size &= 64 \\ ff\_chunks &= 10 \\ sequence\_length &= 24576 \end{aligned} \tag{1}$$

The Wiki dataset was used for these experiments.

1) Model Dimension: This is the dimensions of the attention mechanism d_{model}. We test models ranging from $d_{model} = 512$ to $d_{model} = 1536$. As seen in Fig. 7, it scales linearly with d_{model} and memory usage is under $6GiB$ for $d_{model} = 1536$.

2) Model Depth: Depth refers to the number of layers (attention sub-layer with feed-forward sub-layer) in the model. As described in Section I-B, Reformer uses re-

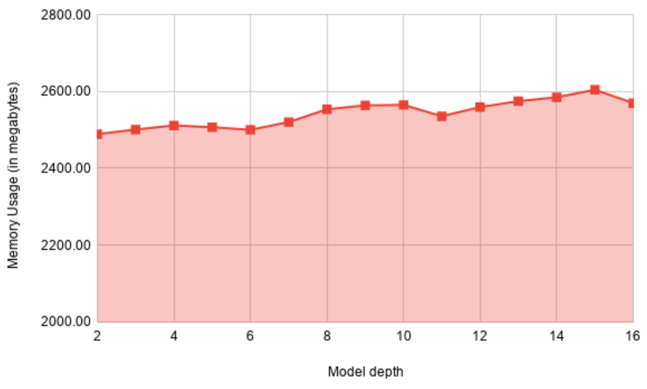


Fig. 7. Memory usage as model depth is varied. There is not much variation or increase in the memory usage owing to reversible layers and weight tying.

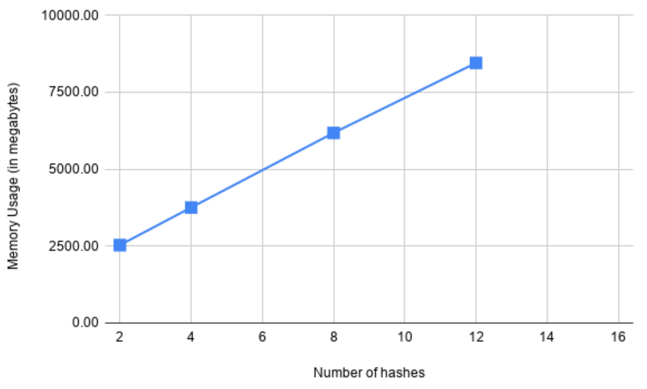


Fig. 8. Memory usage as number of LSH rounds is increased. This has a significant impact on the memory usage. Kitaev et al. [2] suggested 8 rounds of LSH, but that requires a large amount of memory.

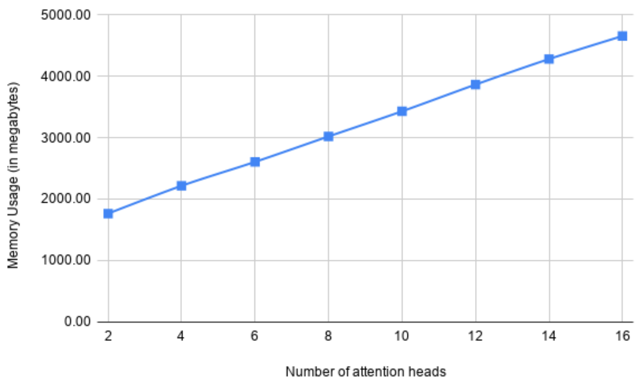


Fig. 9. Effect of number of attention heads on memory usage. Memory usage increases linearly. 8 attention heads can safely be used without using too much memory. [3]

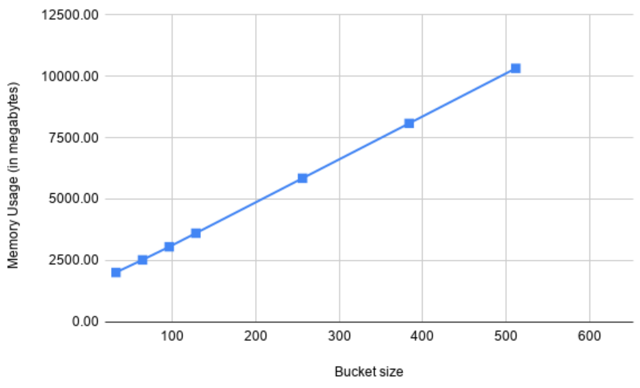


Fig. 10. Memory usage with increasing LSH bucket size (size of QK per bucket). The effect is significant with larger bucket sizes. A value of 64 is used in this paper.

versible layers to make the memory requirements nearly constant as the depth is increased. Fig. 7 shows that the memory usage does not increase as the depth is increased. This verifies the effectiveness of reversible layers.

3) Number of hashing rounds: LSH does not always guarantee that similar vectors fall in the same bucket. Thus, multiple rounds of LSH are applied in parallel and the mode of that is taken as the hash. More number of hashing rounds leads to better accuracy but the memory requirements can increase significantly. Fig. 8 shows the substantial increase in memory usage as number of hashing rounds n_{hashes} is increased. Using $n_{hashes} = 8$ requires over $6GiB$ of memory even for a small model (refer 1).

4) Number of attention heads: Reformer (and Transformer) uses multi-headed attention to get different representations of the same sequence. Generally, higher the number of attentions heads, higher the accuracy. Fig. 9 shows a linear increase in memory usage as number of heads is increased. The increase is significantly smaller than that of n_{hashes}, hence we can use a higher values of n_{heads} without increasing the model's memory requirements too much.

5) Bucket size: As described in Section I-B, LSH Attention clusters inputs into different buckets. Attention is applied inside each of the buckets and then between buckets. Higher bucket size will lead to lesser number of buckets and thus more dot product computation which brings it closer to full attention. Increasing bucket size has a substantial effect on the memory usage as seen in Fig. 10. It grows exponentially and leads to over $10GiB$ of memory usage on a bucket size of 512 which implies a single bucket. Thus, we use a small bucket size of 64.

6) Batch size: As in all deep neural networks, we use batch training in ReformerLM to train on multiple sequences at a time. Generally, batch size has a very significant affect on the memory usage and it is the first hyper-parameter to train when running into memory issues. Though in this case, as shown in this section, other hyper-parameters can also have a very significant effect on the memory usage. Fig. 11 shows the linear increase in memory usage and training time (per iteration) as the batch size is increased.

[3]The model dimensions had to be changed very slightly such that it was divisible by the number of heads.

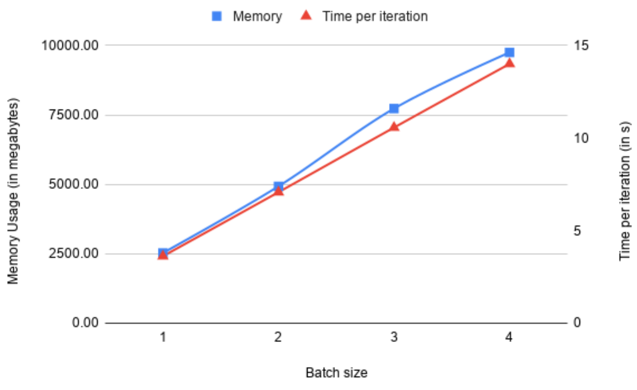


Fig. 11. Memory usage and time taken per iteration with increasing batch size. Clearly, batch size is the most important parameter when optimising for memory. Though, other parameters can influence memory requirements substantially.

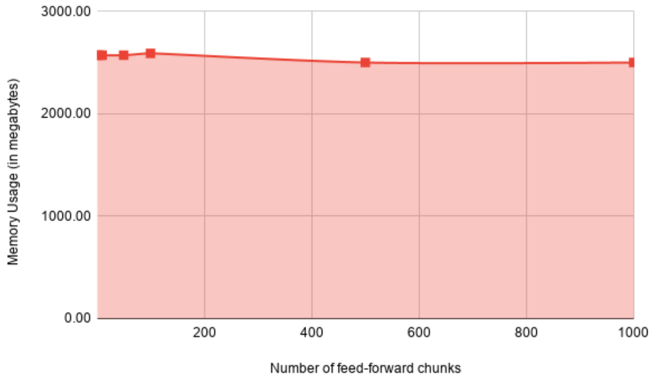


Fig. 12. Memory usage with varying number of feed-forward chunks. It can be seen that this parameter does not influence the memory requirements in any substantial manner. [5]

7) *Feed-forward Chunking:* Feed-forward chunking was described in section I-B. It should theoretically reduce the memory requirements significantly, but in our testing we could not find any improvements when the number of chunks was changed. Fig. 12 shows this effect, there is no significant change as the number of chunks is varied.

8) *Sequence Length:* Sequence length is the maximum number of tokens in the input sequence. Traditionally Transformers can only process upto a few hundreds or few thousands of tokens using a single accelerator. The LSH Attention in Reformer is the most important technique for efficient processing on longer sequences. Thus, we compare a ReformerLM with LSH Attention to a ReformerLM with full (dot product) attention on a range of sequence lengths. Fig. 13 shows the results from this testing. It is evident that LSH Attention makes the time complexity of the model linear in terms of sequence length and can process even $81,920$ tokens on a single GPU. On the other hand, the model with full attention can only process upto $8,192$ tokens at a time on the same machine. The

[5]A ReformerLM with depth 16 was used for this experiment.

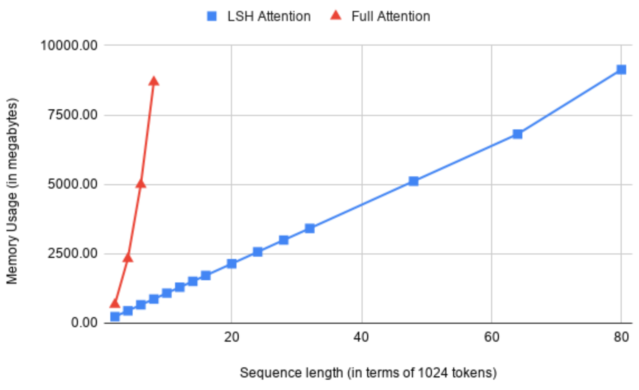


Fig. 13. Memory usage with varying input sequence length of a complete ReformerLM model compared to a ReformerLM model with full attention (instead of LSH attention). The standard ReformerlM scales linearly with sequence length and can even fit $81,920$ tokens on a single GPU[7], while the full attention model runs out of memory for sequences over $8,192$ in length and scales exponentially.

time complexity of the full attention model seems to be exponential in terms of sequence length. This shows the efficiency of Reformer.

9) *Reversible Layers:* Reformer uses reversible layers to reduce the memory requirements of a deep model (a model with large number of layers). With reversible layers, the memory requirement is reduced such that multiple layers do not require additional memory. This is validated in our testing as seen in Fig. 14. With Reversible layers, the memory requirements is constant as the number of layers is increased, while the model without reversible layers requires more memory (increases linearly) as the depth is increased.

Fig. 15 shows the comparison between the same two models but with varying sequence length. Again we see that the Refomer model with reversible layers has a much lower memory footprint across all sequence lengths. The model without reversible layers can only process upto $20,480$ on the same machine. This emprically proves the effectiveness of reversible layers in reducing memory requirements. Comparing with Fig. 13, it can inferred that LSH Attention has a higher effect on the time complexity (with respect to sequence length) than reversible layers.

C. Reformer LM without pre-training

In this experiment, we train a newly initialised Reformer LM on a subset of the BigPatent dataset for 10 epochs. The model was not able to learn any representation with the amount of data given and training iterations. The model predicted the same token over and over again and scores were all zeros or nearly zero.

D. Pre-trained Reformer LM using GSG

A Reformer LM is first pre-trained, using GSG, on a subset of the Wiki corpus for several epochs. Pre-training

[7]A ReformerLM with depth 16 was used for this experiment. Kitaev et al. [2] have demonstrated Reformer processing over $500,000$ tokens on a single accelerator.

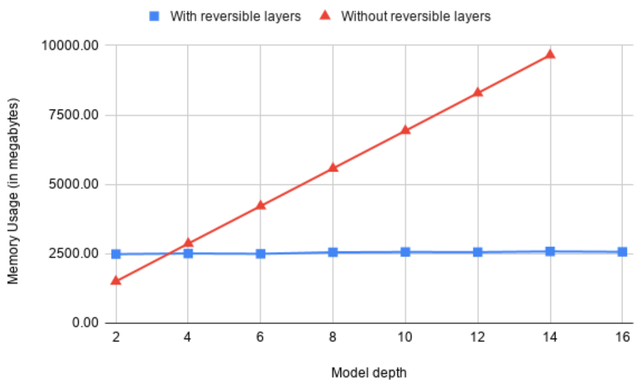


Fig. 14. Memory usage with varying model depth of a complete ReformerLM model compared to a ReformerLM model with reversible layers disabled. Disabling reversible layers makes the model scale linearly (in memory) with number of layers since more memory is required to store activations at each layer.

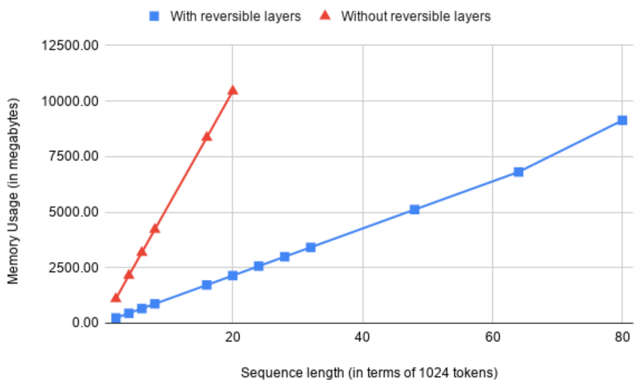


Fig. 15. Memory usage with varying input sequence length of a complete ReformerLM model compared to a ReformerLM model with reversible layers disabled. The model without reversible layer also scales linearly but uses much more memory and can process only upto 20, 480 tokens before running out of memory.

helps it learn internal representations for words and GSG helps approximately learn to construct a summary. The pre-trained model is then fine-tuned on a subset of the Big-Patent summarisation dataset. The trained models were then evaluated using metrics such as BLEU and ROUGE. The evaluation results are summarised in Table I. It can be noticed that non-pre-trained model did not learn anything while the pre-trained model achieved respectable scores for such little data and training time.

E. Examples of generated summaries

A sample generated from the model after fine-tuning it on a very small subset of BigPatent is given below.

> whre intersulural trestle linearly oxes is sensed be adiable to brashing areas an used by a ground by caross . as an ierce increased be container for device includes a water surface , is achieved by mounted rods of supporting a tube of directed br is supper rod into a briecting the bioccle of a ground in plate , a peump of a drose oil fortion , and estruce , escure outface clitable by products the its of polycomic emplements , secured on a plurality of a ultrail clot alloward to in condition in

TABLE I
EVALUATION RESULTS FOR PRE-TRAINED AND NON-PRE-TRAINED REFORMER LM

Metric	Pre-Trained	Non-Pre-Trained
BLEU-1	22.71	0.0
BLEU-2	7.86	0.0
BLEU-3	2.86	0.0
BLEU-4	6.31	0.0
ROUGE-1	25.36	0.48
ROUGE-2	2.79	0.0
ROUGE-3	0.50	0.0
ROUGE-4	0.0	0.0
ROUGE-L	21.62	0.98
ROUGE-W	5.87	0.20

The summary is far from perfect, but it is able to generate English sentences with some spelling mistakes; this is quite good for a model that was trained on a very few data and for a short time. This demonstrates the efficiency of Reformer, a relatively large reformer model was trained on long sequences (24K) with limited resources and data.

V. CONCLUSIONS AND FUTURE WORK

This paper demonstrates the efficiency of Reformer in terms of memory and time requirements and the sample efficiency of pre-training models. GSG is used as a pre-training objective that resembles summarisation more closely compared to other pre-training techniques; this assists the model in fine-tuning stage to achieve summarisation more quickly and accurately. Non-pre-trained models perform very poorly in direct summarisation task when samples and training time are limited. Thus we show that pre-training a decoder-only Reformer LM using GSG can perform abstractive summarisation well when there is a resource constraint. We also describe an efficient implementation of GSG which is optimised using memoization.

Future work related to this paper may include the following.

1) Pre-training a larger Reformer LM on extremely large corpora such as Colossal Clean Crawled Corpus (C4) [10] and HugeNews.
2) Fine-tuning GSG pre-trained Reformer LM on other summarisation datasets such as CNN/Daily Mail [20], XSum [21], Reddit TIFU [22].
3) Building larger Reformer with architectures akin to BERT [4] and T5 [10] and then pre-training them.
4) Using GSG for pre-training other state-of-the-art summarisation models or other transformer models.

REFERENCES

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan,

and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[2] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020.

[3] Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. The reversible residual network: Backpropagation without storing activations, 2017.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[5] Urvashi Khandelwal, Kevin Clark, Dan Jurafsky, and Lukasz Kaiser. Sample efficient text summarization using a single pre-trained transformer. *CoRR*, abs/1905.08836, 2019.

[6] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization, 2019.

[7] Tian Shi, Yaser Keneshloo, Naren Ramakrishnan, and Chandan K Reddy. Neural abstractive text summarization with sequence-to-sequence models. *arXiv preprint arXiv:1812.02303*, 2018.

[8] Colin Raffel and Daniel P. W. Ellis. Feed-forward networks with attention can solve some long-term memory problems, 2015.

[9] Yang Liu and Mirella Lapata. Text summarization with pre-trained encoders. *arXiv preprint arXiv:1908.08345*, 2019.

[10] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[11] Sandeep Subramanian, Raymond Li, Jonathan Pilault, and Christopher Pal. On extractive and abstractive neural document summarization with transformer language models. *arXiv preprint arXiv:1909.03186*, 2019.

[12] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

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

[14] WikiMedia. Wikipedia english articles dump, 2020.

[15] Eva Sharma, Chen Li, and Lu Wang. Bigpatent: A large-scale dataset for abstractive and coherent summarization. *arXiv preprint arXiv:1906.03741*, 2019.

[16] Max Grusky, Mor Naaman, and Yoav Artzi. Newsroom: A dataset of 1.3 million summaries with diverse extractive strategies. *arXiv preprint arXiv:1804.11283*, 2018.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[18] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA, 2002. Association for Computational Linguistics.

[19] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[20] Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend, 2015.

[21] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[22] Byeongchang Kim, Hyunwoo Kim, and Gunhee Kim. Abstractive summarization of reddit posts with multi-level memory networks, 2018.