GENERATING WIKIPEDIA BY SUMMARIZING LONG SEQUENCES

Peter J. Liu*, **Mohammad Saleh***,
Etienne Pot†, **Ben Goodrich, Ryan Sepassi, Łukasz Kaiser, Noam Shazeer**
Google Brain
Mountain View, CA
{peterjliu,msaleh,epot,bgoodrich,rsepassi,lukaszkaiser,noam}@google.com

ABSTRACT

We show that generating English Wikipedia articles can be approached as a multi-document summarization of source documents. We use extractive summarization to coarsely identify salient information and a neural abstractive model to generate the article. For the abstractive model, we introduce a decoder-only architecture that can scalably attend to very long sequences, much longer than typical encoder-decoder architectures used in sequence transduction. We show that this model can generate fluent, coherent multi-sentence paragraphs and even whole Wikipedia articles. When given reference documents, we show it can extract relevant factual information as reflected in perplexity, ROUGE scores and human evaluations.

1 INTRODUCTION

The sequence-to-sequence framework has demonstrated success in natural-language sequence transduction tasks such as machine translation. More recently, neural techniques have been applied to do single-document, abstractive (paraphrasing) text summarization of news articles (Rush et al. (2015), Nallapati et al. (2016)). In this prior work, the input to supervised models ranged from the first sentence to the entire text of an article, and they are trained end-to-end to predict reference summaries. Doing this end-to-end requires a significant number of parallel article-summary pairs since language understanding is a pre-requisite to generate fluent summaries.

In contrast, we consider the task of multi-document summarization, where the input is a collection of related documents from which a summary is distilled. Prior work has focused on extractive summarization, which select sentences or phrases from the input to form the summaries, rather than generating new text. There has been limited application of abstractive neural methods and one possible reason is the paucity of large, labeled datasets.

In this work, we consider English Wikipedia as a supervised machine learning task for multi-document summarization where the input is comprised of a Wikipedia topic (title of article) and a collection of non-Wikipedia reference documents, and the target is the Wikipedia article text. We describe the first attempt to abstractively generate the first section, or *lead*, of Wikipedia articles conditioned on reference text. In addition to running strong baseline models on the task, we modify the Transformer architecture (Vaswani et al., 2017) to only consist of a decoder, which performs better in the case of longer input sequences compared to recurrent neural network (RNN) and Transformer encoder-decoder models. Finally we show our modeling improvements allow us to generate entire Wikipedia articles.

*Joint first-authors. Ordered randomly.
†Work done as a member of the Google Brain Residency (g.co/brainresidency)

2 RELATED WORK

2.1 OTHER DATASETS USED IN NEURAL ABSTRACTIVE SUMMARIZATION

Neural abstractive summarization was pioneered in Rush et al. (2015), where they train headline generation models using the English Gigaword corpus (Graff & Cieri, 2003), consisting of news articles from number of publishers. However, the task is more akin to sentence paraphrasing than summarization as only the first sentence of an article is used to predict the headline, another sentence. RNN-based encoder-decoder models with attention (seq2seq) perform very well on this task in both ROUGE (Lin, 2004), an automatic metric often used in summarization, and human evaluation (Chopra et al., 2016).

In Nallapati et al. (2016), an abstractive summarization dataset is proposed by modifying a question-answering dataset of news articles paired with story highlights from Daily Mail and CNN. This task is more difficult than headline-generation because the information used in the highlights may come from many parts of the article and not only the first sentence. One downside of the dataset is that it has an order-of-magnitude fewer parallel examples (310k vs. 3.8M) to learn from. Standard seq2seq models with attention do less well, and a number of techniques are used to augment performance. Another downside is that it is unclear what the guidelines are for creating story highlights and it is obvious that there are significant stylistic differences between the two news publishers.

In our work we also train neural abstractive models, but in the multi-document regime with Wikipedia. As can be seen in Table 1, the input and output text are generally much larger, with significant variance depending on the article. The summaries (Wikipedia lead) are multiple sentences and sometimes multiple paragraphs, written in a fairly uniform style as encouraged by the Wikipedia Manual of Style[1]. However, the input documents may consist of documents of arbitrary style originating from arbitrary sources.

We also show in Table 1 the ROUGE-1 recall scores of the output given the input, which is the proportion of unigrams/words in the output co-occuring in the input. A higher score corresponds to a dataset more amenable to extractive summarization. In particular, if the output is completely embedded somewhere in the input (e.g. a wiki-clone), the score would be 100. Given a score of only 59.2 compared to 76.1 and 78.7 for other summarization datasets shows that ours is the least amenable to purely extractive methods.

2.2 TASKS INVOLVING WIKIPEDIA

There is a rich body of work incorporating Wikipedia for machine learning tasks, including question-answering (Hewlett et al. (2016), Rajpurkar et al. (2016)) and information extraction (Lehmann et al., 2015), and text generation from structured data (Lebret et al., 2016).

The closest work to ours involving generating Wikipedia is Sauper & Barzilay (2009), where articles are generated extractively (instead of abstractively in our case) from reference documents using learned templates. The Wikipedia articles are restricted to two categories, whereas we use all article types. The reference documents are obtained from a search engine, with the Wikipedia topic used as query similar to our search engine references. However we also show results with documents only found in the References section of the Wikipedia articles.

2.3 TRANSFORMER MODELS

Previous work on neural abstractive summarization relies on RNNs as fundamental modules, mirroring techniques successful in machine translation (MT). Recently, state-of-the-art MT results were obtained using a non-recurrent architecture, called the Transformer (Vaswani et al., 2017). The lack of recurrence enables greater within-training-example parallelization, at the cost of quadratic complexity in the input sequence length. We find the Transformer transfers well to medium length, input sequence summarization and describe modifications to better handle longer sequences.

[1] https://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style

Table 1: Order of magnitude input/output sizes and unigram recall for summarization datasets.

Dataset	Input	Output	# examples	ROUGE-1 R
Gigaword (Graff & Cieri, 2003)	10^1	10^1	10^6	78.7
CNN/DailyMail (Nallapati et al., 2016)	10^2–10^3	10^1	10^5	76.1
WikiSum (ours)	10^2–10^6	10^1–10^3	10^6	59.2

Table 2: Percentiles for different aspects of WikiSum dataset. Size is in number of words.

Percentile	20	40	50	60	80	100
Lead Size	37	62	78	98	166	10,034
Num Citations	1	2	2	3	5	1,029
Citations Size	562	1,467	2,296	3,592	10,320	6,159,463
Num Search Results	10	20	26	31	46	2,095
Search Results Size	1,1691	33,989	49,222	68,681	135,533	5,355,671

3 English Wikipedia as a multi-document summarization dataset

Wikipedia, being an encyclopedia, can be viewed as a collection of summaries on various topics given by their title, e.g. "Canada" or "Machine Learning". The source material to be summarized can be viewed as all reputable documents on the Web or books; however, to make the problem more tractable we consider the following subsets of all documents, D:

1. Cited sources: A Wikipedia article that conforms to the style guidelines should be well-supported by citations found in the *References* section of Wikipedia articles. For each article, a_i, we extract all text without markup from crawlable citation documents, $C_i \subset D$, to use as input to our method.
2. Web Search results: To expand the collection of reference documents, we crawl the search results from the Google search engine, using the article section titles as queries. For each query, we collect 10 result pages. From this collection we remove the Wikipedia article itself, which is often among the top results. We also remove "clones", which are detected when there is a high-level of unigram overlap with the article (details provided in A.2.1). We denote these refined search results for an article, a_i, as $S_i \subset D$. Similar to C_i, we extract only the text to use as input.

Table 2 describes overall properties of our WikiSum dataset. Many articles have few citations, motivating our supplementation of the source documents with web search results. On the other hand, citations when available, tend to be of higher-quality. When counting the total words in the entire dataset, it is orders-of-magnitude larger than previous summarization datasets.

To have consistent train/development/test data across corpus-comparison experiments, we restrict the articles to those with at least one crawlable citation. We divide the articles roughly into 80/10/10 for train/development/test subsets, resulting in 1865750, 233252, and 232998 examples respectively.

4 Methods and Models

Because the amount of text in input reference documents (C_i, S_i) can be very large (see Table 2) it is infeasible to train an end-to-end abstractive model given the memory constraints of current hardware. Hence, we first coarsely select a subset of the input using extractive summarization. The second stage involves training an abstractive model that generates the Wikipedia text while conditioning on this extraction. This two-stage process is inspired by by how humans might summarize multiple long documents: First highlight pertinent information, then conditionally generate the summary based on the highlights.

4.1 EXTRACTIVE STAGE

We investigate three extractive methods from the summarization literature, along with a trivial and cheating method, to assess the importance of this stage. For each article, a_i we create a ranked list of paragraphs, $\{p^i_{R_i(j)}\}$, occurring in (C_i, S_i) where $R_i(j)$ is the rank of the jth paragraph p^i_j of (C_i, S_i). From this we select the first L tokens as input to the second abstractive stage.

1. *Identity*: As a trivial baseline extractor, we simply use the first L tokens of the input.
2. *tf-idf*: A non-trivial ranking is to consider ranking paragraphs as documents in a query-retrieval problem, where the query is the title of the article, $T(a_i)$. We compute tf-idf (Ramos et al., 2003) for the query, with respect to the documents, $\{p^i_j\}$. That is, we summate for each word in the query
$$N_w \cdot log(\frac{N_d}{N_{dw}})$$
where N_w, N_d, and N_{dw} are the count of the word in the document, total number of documents, and total number of documents containing the word, respectively.
3. *TextRank* (Mihalcea & Tarau, 2004): A weighted graph is defined where text units are nodes and edges are defined by a similarity measure based on word overlap. An algorithm similar to PageRank (Page et al., 1999) is then used to compute the ranking of text units. We used paragraphs for the text units.
4. *SumBasic* (Nenkova & Vanderwende, 2005): Word frequencies in the input text are used to assign scores to words, which are in turn used to score sentences. After selecting the best scoring sentence, words in it have their scores reduced, and the process is repeated until the desired summary length is reached.
5. *Cheating* To further demonstrate the quality of extraction on the final performance, we implement a cheating extractor that ranks $\{p^i_j\}$ using recall of bigrams in the ground truth text:
$$d(p^i_j, a_i) = \frac{bigrams(p^i_j) \cap bigrams(a_i)}{bigrams(a_i)} \quad (1)$$

4.2 ABSTRACTIVE STAGE

4.2.1 DATA REPRESENTATION

Given the ordered paragraphs $\{p^i_{R_i(j)}\}$, we derive the raw text input simply as the concatenation of the paragraphs in order, the most relevant at the beginning, and prefixed with the title.

We then encode the text using sub-word tokenization similar to Wu et al. (2016) with a vocabulary size of 32,000 yielding tokenized input, x_i:
$$text_i = T(a_i) \| \{p^i_{R_i(j)}\}$$
$$tokenize(text_i) = x_i = (x^1_i, x^2_i, ..., x^{n_i}_i)$$

For various values of L in experiments, we truncate the tokens to form the input sequence:
$$m^L_i = (x^1_i, ...x^{min(L,n_i)}_i)$$

For the output, we use the same vocabulary and tokenization for the Wikipedia lead text but do not do any truncation across experiments.

Next we describe the abstractive models, W, that learn to write articles, $a_i = W(m^L_i)$, which we treat as a sequence transduction problem from very long input sequences (up to $L = 11000$) to medium output sequences (typically less than 500).

4.2.2 BASELINE MODELS

As a baseline we apply the standard LSTM encoder-decoder with attention (seq2seq-att) as in Bahdanau et al. (2014) to this task. As is typical we train to optimize the maximum-likelihood objective:

$$y_i = tokenize(a_i)$$
$$\prod_{i=1}^{N} p(y_i|m_i^L)$$

A stronger, more recent baseline that we use is the non-recurrent Transformer model described in 2.3, which also has symmetric encoder and decoder modules (T-ED).

4.2.3 TRANSFORMER DECODER (T-D)

We introduce a simple but effective modification to T-ED for long sequences that drops the encoder module (almost reducing model parameters by half for a given hyper-parameter set), combines the input and output sequences into a single "sentence" and is trained as a standard language model.

That is, we convert a sequence-transduction example $(m^1, ..., m^n) \mapsto (y^1, ..., y^\eta)$ into the sentence $(w^1, ..., w^{n+\eta+1}) = (m^1, ..., m^n, \delta, y^1, ..., y^\eta)$, where δ is a special separator token and train a model to predict the next word given the previous ones:

$$p(w^1, ..., w^{n+\eta}) = \prod_{j=1}^{n+\eta} p(w^i|w^1, ..., w^{j-1})$$

Since the model is forced to predict the next token in the input, m, as well as y, error signals are propagated from both input and output time-steps during training. We also suspect that for monolingual text-to-text tasks redundant information is re-learned about language in the encoder and decoder. We believe this allows for easier optimization and empirically observe this with longer sequences (see Section 5.3). Note that because of the self-attention of the Transformer, when generating the next token, attention from both m and y are considered. At inference we provide the input sequence, m_i, initially, and auto-regressively generate the output, y_i, as normal.

4.2.4 TRANSFORMER DECODER WITH MEMORY-COMPRESSED ATTENTION (T-DMCA)

To re-use the terminology used to describe the Transformer, the attention is a function of a query (Q) and set of key (K) and value (V) pairs. To handle longer sequences, we modify the multi-head self-attention of the Transformer to reduce memory usage by limiting the dot products between Q and K in:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Local attention: Sequence tokens are divided into blocks of similar length and attention is performed in each block independently. As the attention memory cost per block becomes constant, this modification allow us to keep the number of activations linear with respect to the sequence length. In our experiments, we choose to have blocks of 256 tokens.

Memory-compressed attention: After projecting the tokens into the query, key, and value embeddings, we reduce the number of keys and values by using a strided convolution. The number of queries remains unchanged. This modification allows us to divide the number of activations by a compression factor. In our experiments we use convolution kernels of size 3 with stride 3. In contrast to local attention layers, which only capture the local information within a block, the memory-compressed attention layers are able to exchange information globally on the entire sequence.

These modifications (see Figure 1) allow us in practice to process sequences 3x in length over the T-D model. For both local and memory-compressed attention, masking is added to prevent the queries from attending to future keys and values. Our final architecture is a 5-layer network (LMLML) alternating between local-attention (L) layers and memory-compressed attention (M) layers (in Vaswani et al. (2017) it is 6 identical layers). We also added in some experiments one mixture of experts (MoE) layer (Shazeer et al., 2017) to increase the network's capacity.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.