Faster Transformers for Document Summarization

Vineet Kosaraju
Computer Science
Stanford University
vineetk@stanford.edu

Yap Dian Ang
Electrical Engineering
Stanford University
dayap@stanford.edu

Zaid Nabulsi
Computer Science
Stanford University
znabulsi@stanford.edu

Abstract

The field of sequence transduction has been dominated by complex networks with an encoder/decoder structure. In recent years, the best performing models for various tasks, such as text summarization and machine translation, have included an attention mechanism. The current state of the art is obtained by transformers (8), but these models are extremely inefficient, even intractable, with long inputs. In this paper, we aim to experiment with design architectural improvements to transformers that not only make the use of transformers on long input sequences, but also able to achieve results comparable to the state of the art. In this milestone, we present our initial findings, which includes implementing and running our baseline (the original, unmodified transformer), along with attempts at architectural changes. In the end, we see somewhat promising initial results, giving us confidence in our ability to achieve our overall aim by the end of this research project.

1 Introduction

In recent years, there have been several successful innovations with neural networks that have improved the field of natural language processing. Chief of these innovations are recurrent architectures, such as RNNs (2), which allow for multi-word sequence encodings, and attention mechanisms (8), which improve the accuracy of these recurrent systems. The main drawback of recurrent systems is that they require sequential processing of the input. Transformers aimed to mitigate this limitation of RNNs by replacing the recurrent networks with feedforward layers. Transformers have become highly successful on a wide variety of tasks such as machine translation (8), document summarization (3), language modeling (5), and question answering (1). However, the main drawback with transformers is that their attention layers serve as bottlenecks. Specifically, the attention of transformers is quadratic in performance with respect to the input length: $O(\text{len(seq)}^2)$. Thus, this motivates us to design architectural improvements to transformers for more efficient training, while maintaining comparable accuracy to the existing state-of-the-art methods on document summarization.

2 Approach

2.1 Task Definition

The specific task we are focused on is that of automatically summarizing long sentences, paragraphs, and even entire documents. As previously mentioned, the summarization task is one example of several goals in NLP where being able to process large amounts of text efficiently is crucial, due to the long input sequences. Specifically, when summarizing documents with thousands of words, many traditional language models that work well, such as RNNs and transformers, become computationally inefficient due to massive memory and compute requirements. We utilize the WikiHow Large Scale Text Summarization dataset (Section 3.1) and focus on summarizing a full document into a single paragraph. Notation wise, for a single example we map the raw text, $(x_1, x_2, \ldots, x_{m'})$, to a summarized version, $(y_1, y_2, \ldots, y_{n'})$, where $m' >> n'$ and $m', n' \in \mathbb{Z}$, with x_i, y_j as word tokens.

2.2 Transformers

For our baseline architecture we chose a vanilla transformer. As part of its core architecture, each transformer is composed of several layers of encoder and decoder modules (generally $N = 6$ of each). As opposed to using a recurrent structure to sequentially process words, the transformer parallelizes the computation using several linear (fully connected) layers and self-attentive layers (Appendix: Figure 2). Within each encoder/decoder is also residual skip-connections to aid the learning process.

Each attention layer is slightly more complex and takes in three values: a value (V), a key (K), and a query (Q). From these values, it applies a standard dot-product attention, as shown in Equation 1, with a scaling factor of $\frac{1}{\sqrt{d_k}}$ added to ensure that the value of the dot product doesn't grow too large in magnitude with respect to d_k, the dimension of the key.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

However, instead of running self-attention on an inputted key, query, and value only once, the network instead makes use of a module called multi-head attention. This module repeatedly runs attention using different weights on the same inputted triplet. The benefit here is that multiple attentive properties can be learned in a parallel fashion, essentially allowing for ensembling of attention weights and learning of a full distribution of word importance.

Generall, we define $h = 8$ unique heads, where each head has its own learnable weights, ${W_i}^Q$ for the query, ${W_i}^K$ for the key, and ${W_i}^V$ for the value, along with an overall set of learnable weights, W^O:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{2}$$

2.3 Proposed Models

While the core transformer model is very promising and improves upon recurrent architectures, it does have some limitations. Consider Table 1 (Appendix), where we compare the transformer model's performance in terms of asymptotic runtime, number of operations, and dependency length. Note that n represents the length of the sequence, and d represents the embedding hidden dimension.

While its true that the transformer improves over a recurrent architecture by reducing the number of sequential operations from n to 1, this improvement does have some limitations. Specifically, the asymptotic runtime per operation of the transformer is only faster than recurrence when $n < d$. This is generally true in NMT tasks, where translated sentences are relatively short, but is not true in summarization, where n is often the entire length of a document.

The main reason this runtime is large in the transformer is that it is supporting a maximum dependency length of $O(n)$: in other words, the very first word can have dependencies on the very last word. Again, this is useful in NMT, but is not as necessary in document summarization, where we often want to summarize paragraphs individually into shorter sentences. As such, in our proposed changes we hope to decrease the runtime of each operation, while maintaining $O(1)$ operations, by leveraging the reduced need for long dependencies in document summarization tasks.

Convolutional Model As the dot-product attention remained the largest bottleneck in the transformer architecture, we wanted to reduce the number of computations it required. To do so, we shortened the input to that layer by introducing a convolution beforehand and a deconvolution afterwards (Figure 1). The convolutions would group surrounding words into phrases, and using a nonzero stride would reduce the size of the input. We propose two ways of incorporating this change into the multi-head attention: by using the same convolution across all the heads ("Convolutional Attention"), or by introducing a unique convolution for each head ("Multi-Head Conv-Attn"). We suspect the latter will perform more optimally, as it may learn a different grouping of words for each head of the model.

Neighborhood Attention As discussed earlier, in summarization we no longer need long range dependencies between words in the entire document. Instead, usually dependencies only occur within a sentence, or within a paragraph. To explicitly allow for these shorter-term dependencies, we implement a neighborhood attention model that only attends to the surrounding $2r + 1$ words in the sentence (r is the max distance from the target word in the neighborhood). This is similar to the restricted attention proposed but not tested by Vaswani et. al (8).

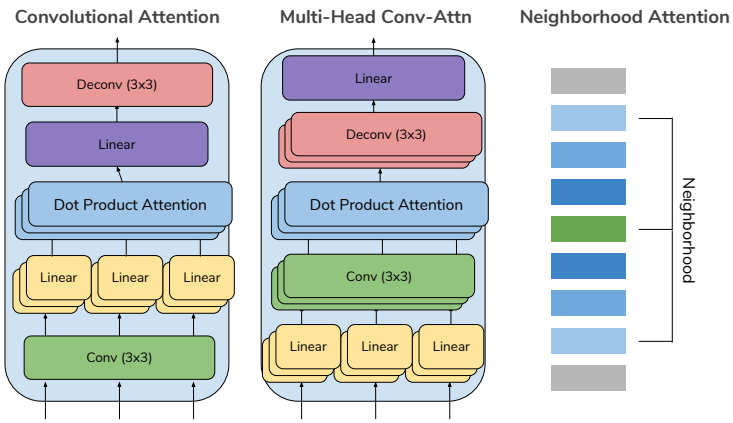

Figure 1: We propose two main modifications to the attention architecture. In the first, we introduce convolutions and deconvolutions before and after the attention to group words into phrases and reduce the operations the attention performs. This can be done either globally (left), or once per each head (middle). We also propose a neighborhood attention that only looks at the $2r + 1$ surronding words.

3 Experiments

3.1 Datasets

We focus on conditional language modelling, specifically single-document summarization using the Wikihow dataset. In the raw dataset, the raw text is the source and the bold text is the summary. We preprocessed the dataset by combining an article's paragraphs into one line, such that `src.txt` and `tgt.txt` match line by line and we have one to one mappings between articles and summaries.

3.2 Experimental Details

We wrote the preprocessing scripts for the dataset and the validation scripts ourselves. Model-wise, we wrote our code for the core sections of the transformer, such as the soft attention module, train and test iterator, the encooder-decoder model architecture, and the main train function with GPU parallelization in Pytorch. Other implementation details which weren't as familiar for us such as positional encoding, multi-head attention, Noam's learning rate decay, and label smoothing were written with guidance from the OpenNMT's Annotated Transformer and Vaswani's paper.

Implementation wise, we use $h = 8$ unique heads and $d_{model} = 512$ for multi-headed attention. We optimized over KL divergence loss between the model output, and the label-smoothed ground truth (Appendix: Equation 3). We used $p_{dropout} = 0.1$ consistently for dropout probabilities, and Noam learning rate decay scheme (Appendix: Figure 3) with 2000 warm up steps and Adam optimizer with $\beta_1 = 0.9, \beta_2 = 0.98$ and initial learning rate of 1. Building on top of the baseline, in the memory compressed attention, we applied filters of size 3 and stride 3, and $r = 100$ for the neighborhood size.

In the positional encoding module, we add positional encodings with sinusoidal functions to input embeddings before feeding into the encoder and decoder modules, as shown below:

$$PE_{pos,2i} = \sin\left(\frac{pos}{1000^{2i/d_{model}}}\right), PE_{pos,2i+1} = \cos\left(\frac{pos}{1000^{2i/d_{model}}}\right)$$

We faced several difficulties when running our baseline model: since our summarization task takes in a long sequence of text as inputs, and the baseline model attends to every single word in the source text, we ran into CUDA out of memory issues even with Azure GPU devices. This issue was the prime motivation for us to improve our baseline and making transformers faster, since the bottleneck lies in the attention module in very long sequences. To allow our baseline to run, we only consider shorter articles with maximum length of 1000 tokens and batch size of 512 to allow the model to fit into memory, and we hypothesize that with our proposed model of memory-compressed attention with convolutions, we can reduce this memory and time bottleneck on longer sequences. The model was trained for 10 epochs on a Tesla K80.

3.3 Evaluation Method

In order to evaluate our models and to establish a basis for comparison, we use three main methods of evaluation: speed (as measured in time per epoch), ROUGE, and perplexity. We measure speed as one of our goals of our research is to increase the efficiency of existing approaches. ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a metric that measures co-occurrence statistics (7). There are numerous different flavors of ROUGE that we utilize, including ROUGE-N with $N = 2$ and ROUGE-L-F1. (6). ROUGE-N is a measure of the overlap of N-grams between the system and reference summaries, and is given by:

$$\text{ROUGE-N} = \frac{\sum_{C \in RSS} \sum_{gram_n \in C} COUNT_{match}(gram_n)}{\sum_{C \in RSS} \sum_{gram_n \in C} COUNT(gram_n)}$$

where RSS is a set of reference summaries-reference summary set, $COUNT_{match}(gram_n)$ is the maximum number of n-grams co-occurring in a candidate summary and a reference summary and $Count(gram_n)$ is the number of n-grams in the reference summary.

We also utilize ROUGE-L-F1, which is a longest common subsequence measurement that takes into account sentence level structure similarity and identifies the longest co-occurring sequence n-grams. We use this type of ROUGE (especially the F1 type) as it is more appropriate in this setting, since we do not want to explicitly constrain the outptut length. (4) Furthermore, this metric is used commonly in text summarization, allowing us to effectively compare our results with other approaches.

3.4 Results

We begin by first reporting the speeds of the baseline, along with our models. For consistency, and to allow for a better basis of comparison, we report token processing speed, as measured in tokens per second. Loss curves and attention visualizations are in the Appendix.

	Training Phase	Evaluation Phase
Transformers Baseline	292 tokens/second	1026 tokens/second
Convolutional Model	352 tokens/second	1249 tokens/second
Neighborhood Attention	373 tokens/second	1298 tokens/second

In the table, we see that both our convolutional model and our neighborhood attention model achieve faster speeds than the transformers baseline, which is what we expected because in the baseline model, attention is applied over the entire document, which is the bottleneck. On the other hand, in both the neighborhood model and the convolutional model, we restrict attention size significantly, as discussed in Section 2.3. For each of our models, we also report ROUGE-2 and ROUGE-L-F1 scores, as well as perplexity. The values are found in the table:

	ROUGE-L-F1	Perplexity
Transformers Baseline	12.8	46.2
Convolutional Model	12.7	45.3
Neighborhood Attention	13.1	44.7

The first observation we make is that our results are sub-optimal, even when we use the state of the art transformers, demonstrating the difficulty of the task of text summarization for long inputs. Furthermore, we note that both of the models we introduced achieve very similar (and at times better) metrics as the state of the art transformers baseline does. This gives us confidence in our approach, as we are already challenging the state of the art in the task of text summarization. However, as we can deduce from the numbers, there is still plenty of room for improvement overall, and we look to drastically enhance our model as we progress through the project.

For our final report, we have three main goals that we'd like to accomplish. First, we'd like to experiment with different settings of r for the neighborhood attention, and different kernel sizes and strides for the convolution attentions, to see how those impact timing and performance. We'd also like to combine the two architectural improvements into one unified architecture, to see the impact on runtime. Finally, we'd like to perform a more deep look at the errors our model is making, by analzying the generated tokens, and visualizing attention heatmaps.

4 Additional Information

We are doing a custom project under Guidance from Kevin Clark, without external collaborators and without sharing this project with other classes.

References

[1] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and L. Kaiser. Universal transformers. *CoRR*, abs/1807.03819, 2018.

[2] S. Fernández, A. Graves, and J. Schmidhuber. An application of recurrent neural networks to discriminative keyword spotting. In *Proceedings of the 17th International Conference on Artificial Neural Networks*, ICANN'07, pages 220–229, Berlin, Heidelberg, 2007. Springer-Verlag.

[3] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer. Generating wikipedia by summarizing long sequences. *International Conference on Learning Representations*, 01 2018.

[4] C.-Y. Lin and F. J. Och. Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *ACL*, 2004.

[5] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2018.

[6] N. Schluter. The limits of automatic summarisation according to rouge. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 41–45. Association for Computational Linguistics, 2017.

[7] J. Steinberger and K. Jezek. Evaluation measures for text summarization. *Computing and Informatics*, 28:251–275, 01 2009.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 06 2017.

5 Appendix

In this section we present supplementary figures, tables, and equations that we were unable to present in the main portion of the paper.

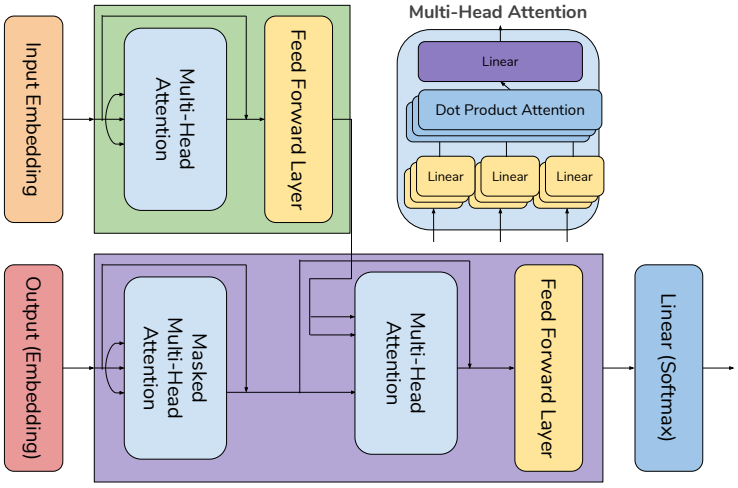

Figure 2: Baseline Transformer Architecture: The vanilla transformer consists of several encoder (green) and decoder (purple) blocks. Inside each block is several residual skip connections. Each block uses the multi-head attention module, which computes self-attention multiple times in parallel.

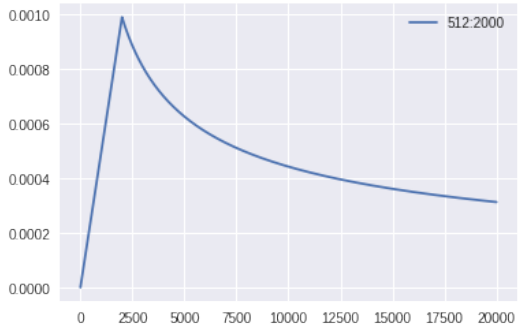

Figure 3: Learning rate scheduler with 2000 warm-up steps, and Adam optimizer with $\beta_1 = 0.9, \beta_2 = 0.98$ and initial learning rate of 1.

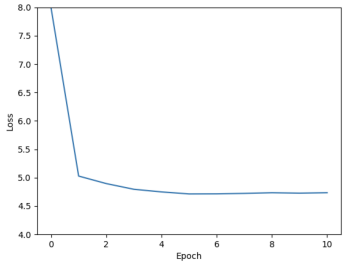

Figure 4: Loss curve for baseline model

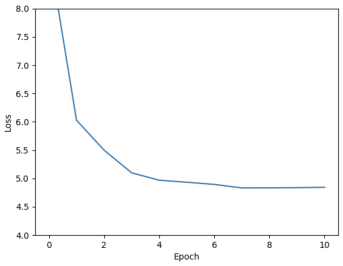

Figure 5: Loss curve for convolutional model

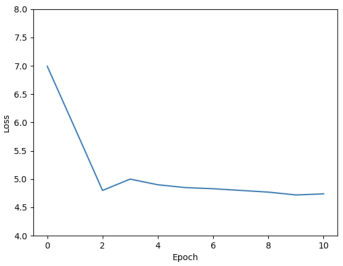

Figure 6: Loss curve for neighborhood model

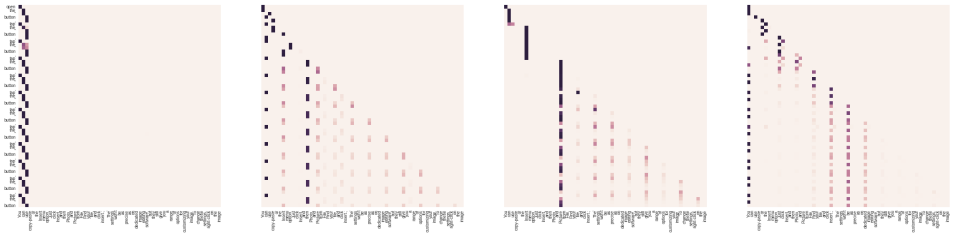

Figure 7: Visualization of attention in decoder level 6 for baseline.

Layer	**Asymptotic Runtime**	**Operations**	**Dependency Length**
RNN (Recurrent)	$O(n \cdot d^2)$	$O(n)$	$O(n)$
ATTN (Transformer)	$O(n^2 \cdot d)$	$O(1)$	$O(n)$

Table 1: Comparison of runtime, number of operations, and length of dependencies supported for a recurrent architecture compared to the transformer architecture.

$$L = \text{KL}(p||q) = \mathbb{E}_{x \sim p(x)} \left[\log \frac{p(x)}{q(x)} \right] \tag{3}$$

$$q = q(1 - \epsilon) + (1 - q) \frac{\epsilon}{|V_t| - 1} \tag{4}$$

$$\text{head}_i = \text{Attention}(QW_i{}^Q, KW_i{}^K, VW_i{}^V) \tag{5}$$