CS224N Project Report
Faster Transformers for Text Summarization

Amaury Sabran
asabran@stanford.edu [*]

Alexandre Matton
alexmt@stanford.edu [*]

Abstract

A recently proposed neural network architecture called the Transformer [1] works very well for many NLP tasks. However, its runtime is quadratic in the length of the input sequence, which means it can be slow when processing long documents or taking characters as inputs. In this project, we provide an in-depth analysis of the Transformer's speed bottlenecks and introduce models that address the bottlenecks of the encoder. We use these models for the task of Text Summarization and evaluate the trade-off between speed and performance. We find that models with strided convolutions applied directly on the input or in the attention layers gives a significant speed-up to the transformer while only sacrificing a small part of the accuracy. Convolutional and local encoders outperform the transformer and it looks like local information plays a big role in text summarization.

1 Introduction

Traditionally, sequence to sequence tasks such as Neural Machine Translation and Text Summarization were addressed with recurrent neural networks (RNN, LSTM) [2]. The encoder RNN would read the input sequence and encode it in a fixed-size vector and the decoder RNN would take this fixed-size encoding as input to produce the output sequence. In order to keep track of the sequence content, the encoder RNN reads the sequence word by word and updates a inner hidden state.

Recent advances in NLP rely on the success of self-attention and source-to-target attention models (fig. 1, 2). The attention mechanism [3] computes attention weights for each input word and produces a weighted average over context representations. Source-to-target attention extracts information from the source sequence to predict the target sequence while self-attention operates only on the current sequence. Most of the latest successes in NMT build on an architecture that uses only self-attention and source-to-target attention, without any recurrent unit: the Transformer [1]. Despite its successes in Neural Machine Translation, the Transformer's running time is quadratic in the length of the input sequence, which makes it hard to use for long input texts such as news articles.

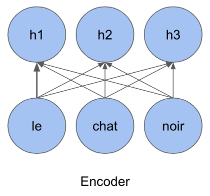

Figure 1: Self-Attention

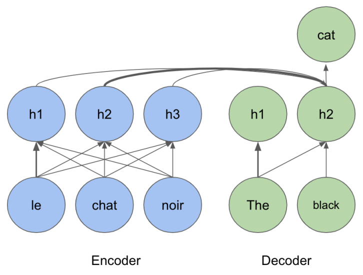

Figure 2: Source-to-target Attention

[*]This project was supervised by Kevin Clark kevclark@cs.stanford.edu

In this project, we design and apply several variants of the Transformer to Text Summarization on the CNN-DailyMail dataset [4] [5]. Even if our GPU capacity does not allow us to compare favorably to state of the art Text Summarization results, we provide an analysis of the speed-performance trade-off of the different models.

We find that for input sequences of length 400-800 (the typical range of article lengths for the CNN-DailyMail dataset), the Transformer is actually pretty fast thanks to GPU optimizations and there is no quadratic trend in the self-attention runtime. Even though our project focused on making the self-attentive encoder faster, we show that for those range of input lengths the Transformer runtime is actually dominated by the decoder speed (due to the output vocab size) and by the cost of large linear projections in the encoder.

We introduce three new models : the **local Transformer**, the **Transformer with input convolutions** and the **Memory Compressed Attention Transformer** and evaluate their Text Summarization performances. Our experiments show that faster variants of the Transformer can be used without losing too much performance. We show that models that focus on local interactions of words perform better than the transformer that extract information from all words at once. The speed and small number of parameters of Lightweight Convolutions allowed us to train it with a bigger architecture during 17 epochs and we obtain a competitive ROUGE-1 F1 score of 38.4.

2 Related Work

2.1 Text Summarization

Abstractive Text Summarization (as opposed to Extractive Text Summariation) was recently made possible thanks to the advance of sequence-to-sequence learning [2]. The sequence-to-sequence RNN of [6] is a simple RNN-encoder decoder with source-to-target attention. They introduce a copy mechanism that allows the decoder to copy a word from the input sentence instead of generating a new word from the vocabulary. The pointer-generator network of [7] is also a sequence-to-sequence model with a bidirectional LSTM encoder and a LSTM decoder. They also use a pointer copy mechanism and source-to-target attention and they introduce a coverage vector that prevents repetition in the summary.

These two models use recurrent encoders and decoders and only one source-to-target attention while our models rely entirely on self-attention and source-to-target attentions with up to 6 layers of self-attention and source-to-target attention.

2.2 Self-attention and the Transformer architecture

The Transformer architecture [1] builds on a base block called the self-attention module. Each module consists in a number H of heads that have different weights and can attend to different positions. For simplicity, we will describe a one-head module and show that its cost is quadratic in the length of the input sequence.

A head takes as input $X \in \mathbb{R}^{n \times d}$ and applies three projections to obtain queries (Q), keys (K) and values (V), where n is the number of input tokens (or time-steps) and d is the input dimension. The module computes dot products between queries and keys, scales the dot products for training stability and applies a softmax to produce attention-scores. The values are combined with a weighted sum according to those attention scores.

The output of one head can be summarized as follows :

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d}})V$$
$$SelfAttention(X) = Attention(XW_Q, XW_K, XW_V)$$

Where the matrices W_Q, W_K and W_V are trainable weights. Here $Q \in \mathbb{R}^{n \times d}$ and $K \in \mathbb{R}^{n \times d}$ hence the matrix product QK^T has a cost $O(d \times n^2)$ which is quadratic in the length of the input sequence. This quadratic cost is especially noticeable for long input-sequences such as documents to be summarized or character-level inputs.

2.3 Faster self-attention

Some solutions have been proposed for the quadratic cost of self-attention.

The Lightweight Convolutions model of [8] keeps the high-level architecture of the Transformer but replaces self-attention by lightweight convolutions. Lightweight convolutions are a mix between convolutions and attention where a weighted average of input representations is done over a sliding window. This architecture has a linear complexity in the length of the input length and requires much less parameters to train than regular attention. They report state-of-the-art or close to state-of-the-art results on several Translation and Text-Summarization tasks.

The Transformer decoder with memory-compressed attention model of [9] generates wikipedia articles ($\sim$summaries) from the reference articles. They propose a new architecture to handle inputs and outputs typically 100 times larger than previous Text-Summarization models. Since the data is monolingual, they remove the encoder and use the concatenation of the articles and the summary as input for the decoder. They also introduce two novel ideas to speed-up transformers. **Local attention** consists in dividing input sequences into blocks of the same length and performing attention in each block independently. **Memory-compressed attention** performs a strided convolution on the keys and values in the attention layers to artificially reduce the number of input-tokens in the attention

3 Approach

As explained above, the complexity of the Transformer is quadratic ($O(n^2 \times d)$). We implemented several models that rely on an encoder-decoder structure very similar to the Transformer. All our modifications were done on the Transformer encoder as this is the where the quadratic complexity happens. The decoder also has self-attention layers with quadratic complexity but the decoder inputs (summaries) are typically much shorter than the encoder inputs (articles).

3.1 Models

Local Transformer (ours) : The local transformer divides the input sequence into chunks of fixed-size which are processed independently by the encoder. With a fixed-size window attention, the cost becomes *linear* while retaining many advantages of self attention.
Indeed, attention windows of size k, which is supposed to be much smaller than n, the local transformer replaces one self-attention on the whole input by n/k self-attentions on k tokens. The global complexity of the new block is $\mathcal{O}(k^2 \times d \times n/k) = \mathcal{O}(n \times k \times d)$. The architecture is represented in Figure 3.

Local Transformer with shifts (ours): One major problem of the local transformer is that it prevents information flow from one chunk to another. This is particularly problematic as words located at the border of a chunks can't see words at the other side of the chunk. One way to fix that is to shift all chunks by half of their size in odd blocks of the transformer. With enough blocks, this makes it possible for information to travel between all parts of the input sequence. Moreover, the global complexity is exactly the same as the one of the local transformer, i.e. $\mathcal{O}(n \times k \times d)$. The shift is showed in Figure 4.

Convolution before Transformer (ours): This method consists in reducing the size of the input before feeding it to the transformer. One way to do that is to run a convolution layer before the Transformer blocks with stride equal to the size of the filters. From a high-level perspective, the convolution summarizes small contiguous groups of words (typically 4) and the transformer processes the summarized inputs.
The complexity of the initial convolution is $\mathcal{O}(n \times d^2)$ and the complexity of the self-attention layers become $\mathcal{O}((n/k)^2 \times d)$, where k is the kernel-size of the convolution layer. Hence the overall complexity becomes $\mathcal{O}(n \times d^2 + (n/k)^2 \times d)$.

Memory-compressed attention (adapted from [9]) : This architecture also uses strided convolutions to decrease the size of the inputs. However, the convolutions are located in the self-attention

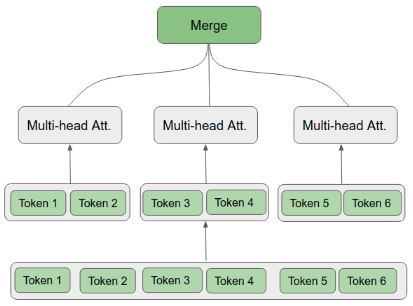

Figure 3: Local Attention (even layers)

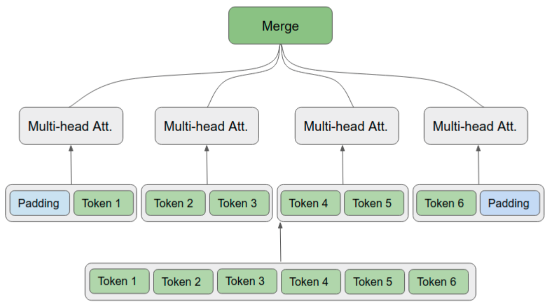

Figure 4: Local Attention with shift (odd layers)

layers. The memory compressed-module is described as follows :

$$CompressedMemoryAttention(Q, K, V) = softmax(\frac{c(Q)c(K)^T}{\sqrt{d}})V$$

Where $Q \in \mathbb{R}^{n,d}, K \in \mathbb{R}^{n,d}$ and the strided convolution outputs with kernel-size and stride k are $c(Q) \in \mathbb{R}^{\frac{n}{k},d}, c(K) \in \mathbb{R}^{\frac{n}{k},d}$. The complexity of each convolution is $\mathcal{O}(n \times d^2)$, and the complexity of the attention computation becomes $\mathcal{O}(\frac{n^2}{k} \times d)$, so the overall complexity of the memory-compressed attention mechanism is $\mathcal{O}(n \times d^2 + \frac{n^2}{k} \times d)$. This architecture is a compromise between the classic Transformer and the one with the convolution on the inputs.

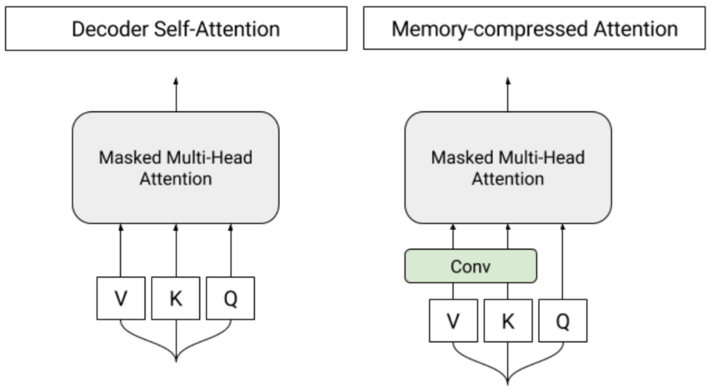
Decoder Self-Attention
Memory-compressed Attention
Masked Multi-Head Attention
Masked Multi-Head Attention
Conv
V
K
Q
V
K
Q

Figure 5: **Left**: Original self-attention **Right**: Memory-compressed attention

Lightweight convolutions (from [8]) : This model replaces self-attention layers by some kind of local convolutions where each filter only takes into account one dimension. This is represented by a matrix $W \in \mathbb{R}^{d \times k}$ with k the kernel size (i.e. size of the window for the convolutions). The general equation is:

$$O_{i,c} = \sum_{j=1}^{k} W'_{c,j} \cdot X_{(i+j-\lceil \frac{k+1}{2} \rceil),c}$$

where $X \in \mathbb{R}^{n \times d}$ is the input and $O \in \mathbb{R}^{n \times d}$ is the output. W' is the matrix W with a softmax layer is applied to each row (i.e. across each channel). This is done to increase training stability, and makes each output token more aware of its context.
The overall complexity of a Lightweight convolution block is $\mathcal{O}(n \times k \times d)$.

3.2 Code

We designed our own data-loading, training and evaluation code and used the Fairseq library [10] for training utils and state-of-the-art implementations of the Transformer and Lightweight convolution models. We built the other models using the same framework and integrated them in our branch of the Fairseq library.

Our code is available on github and can be found here. It uses our branch of the Fairseq repository available here.

4 Experiments

4.1 Data

The CNN-DailyMail Dataset The Text Summarization dataset we used is the CNN-DailyMail Dataset introduced in [4] and [5], which consists of over 280K news articles paired with multi-sentence summaries. The articles are rather long with 39 sentences on average. During training and testing we truncate the articles to 400 tokens and set a limit of 100 tokens for the summary in the training phase and 120 tokens in the testing phase in order to have comparable results with the ones that are computed in [7].

Evaluation method We evaluated our text summarization results with the ROUGE metric [11]. There are several different types of ROUGE. The ROUGE scores that are generally used in the literature are the F1 ROUGE-1 (for unigrams), ROUGE-2 (for bigrams), and ROUGE-l (which finds the longest common sequence between the hypothesis and the given summary). For all our model, we report the ROUGE-1, ROUGE-2 and ROUGE-l scores.

4.2 Experimental details

Architecture We kept a similar architecture for all the models for comparison purpose. Because of our limited amount of GPU credits and time constraints, we halved all hyperparameters of the original Transformer architecture and kept 3 encoder/decoder layers, 4 attention heads per layer, and a embeddings with 256 dimensions, and 1024 for the hidden states. In the tables, those architectures are referred to as **small**. We used an input and output vocabulary of 50k words.

For the LightConv model we used kernel sizes of 15,31 and 31 for the three encoder layers. For the Local Transformer, we used blocks of 50 tokens. For the Transformer with input convolution we applied a strided convolution with kernel-size 4 and stride 4 to the inputs. For the Memory compressed attention we applied a strided convolution with kernel-size 4 and stride 4 to the keys and values.

Optimizer, loss and regularization We used the adam optimizer [12] with betas (0.9, 0.98). We used the label-smoothed cross entropy loss with parameter $\epsilon = 0.1$ described as

$$\mathcal{L}(y_{true}, \hat{p}) = -(1-\epsilon)\log(\hat{p}(y_{true})) - \epsilon \sum_{y \in Vocab} \log \hat{p}(y)$$

We use an inverse square-root learning rate scheduler with a maximum learning rate of $5 \cdot 10^{-5}$ and a linear warmup of 4000 steps. We use a dropout rate of 0.1 and a weight-decay rate of 10^{-4} for regularization.

Training We trained all our models for 300k iterations with batch size 16, which corresponds to 17 epochs on the CNN-DailyMail dataset. For each model, we report the ROUGE scores for the iteration with the best validation loss. Most of our models did not fully converge in 300k iterations. Unfortunately, we did not have the GPU time and credits to perform longer training.

Generation We use beam-search with beam-size 4 to generate summaries for the test articles and we force the model not to repeat 3-grams since text summarization models tend to repeat themselves.

Speed analysis setup To measure the speed of the models we created dummy datasets with randomly created texts of size ranging from 50 to 2000 tokens. This range contains the typical size of articles from CNN/DailyMail dataset (400) and the size of bigger articles such as the ones used in [9]. We measured the speed of the full models and not the small ones we trained as we considered that we would get more accurate and consistent results. We use batch of 16 sentences on a Tesla K80 GPU.

4.3 Results

4.3.1 ROUGE scores

We present here the ROUGE scores we obtained for all our small architectures after 17 epochs. The scores for the standard (6-layers encoder) Transformer and Lightweight architectures after 17 epochs are present as a reference but are not included in the comparison. The speedup column is the relative speed compared to the transformer architecture, for input sequences of 400 tokens.

Model	ROUGE-1	ROUGE-2	ROUGE-l	Speedup
TransformerSmall	32.39	8.78	26.8	1
LightConvSmall	**36.27**	**14.31**	**30.91**	1.08
TransformerLocalSmall	35.53	14.01	30.62	1.13
TransformerLocalSmall + Shift	35.8	14.54	30.92	1.13
TransformerSmall + Input convolution	30.30	8.63	26.05	1.62
Memory compressed attention	31.43	7.70	26.12	1.01
LightConv	38.37	16.20	32.7	
Transformer	25.55	5.08	22.5	

Table 1: ROUGE scores and speedups for our models

4.3.2 Model comparison

Surprisingly,with our training budget the small transformer gave much better results than the transformer with its traditional architecture. This is probably due to the fact that the full transformer needs more time to converge as it has more parameters.

The TransformerSmall + Input convolution and Memory compressed attention models have performances slightly lower than the TransformerSmall model, which is expected. The drop in performance is not that big, considering the speedup (especially for TransformerSmall + Input convolution).

LightConvSmall and TransformerLocalSmall have better scores than the TransformerSmall and are faster to train. This can be explained since LightConv and TransformerLocal explicitely force the model to compare words that are close while the Transformer model has to learn the idea of relative position only with the positional embeddings.

The block-shift to allow information flow in transformer local does not add much, its effect should be more noticeable for bigger architectures (6 layers).

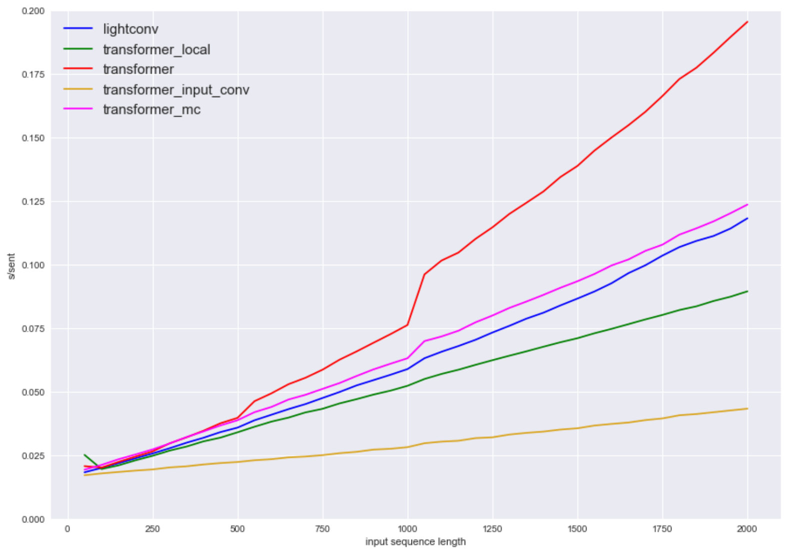

Figure 6: Inverse speed of the different models with different sequence lengths (seq/s)

4.3.3 Speed Experiment

We use our fake datasets of articles of length 50 to 2000 and summaries of length 100 to compute the speed of all our models. The results are displayed in fig. 6, in number of second per processed sentence.

As we show in section. 5.2, for relatively small sequence the quadratic cost of the self-attention layer is small compared to the linear costs of all the other components. Its cost becomes noticeable for sequences of more than 1500 tokens. This is why the speed curves of the all the variants of transformer look mostly linear.

The transformer with input convolution model is by far the fastest (it reduces the length of the input sequence by 4 without losing too much information). Lightconv, TransformerLocal and the transformer with memory compressed attention have similar speeds and offer a significant speedup over the usual transformer architectures for large sequence lengths (around 1300).

We did not find a reasonable explanation for the sudden speed decrease of transformer from sequences to 1000 to 1050 tokens. This might be due to GPU optimizations.

5 Analysis

5.1 Text Summarization and local attention

The main finding of our analysis is that variants of the Transformer architecture that impose local attention (LightConv and TransformerLocal) perform better than the usual Transformer. This result is quite natural since words interact first at a local level and then the meaning of the different sentences interact at the article level.

When we train the usual Transformer architecture on very long texts, the model has to learn that the self-attention should look at close words rather than far-away words uniquely with the positional encodings. This is a hard task, especially for small architectures. With Lightconv and Transformer-Local, we force the model to extract local information when using self-attention. This makes the training much easier and leads to better summarization performance.

For the decoder, the summarization model should be able to look at all the input article at once and we should not use a local source-to-target attention.

5.2 Detailed analysis of the transformer

The transformer has multiple components with different complexities. We computed the runtime of each component by removing them one at a time. The results can be found in the table below, where n is the number of input tokens, m is the number of output tokens. We use the standard transformer architecture from [1] with embeddings of size $d = 512$,hidden layers of size $d' = 2048$ for the MLP at the end of the encoder, a vocabulary of size $V = 50000$ and $N = 6$ blocks in the encoder/decoder.

As expected, the encoder total time grows much more than the decoder total time with n, because of the quadratic self-attention, which takes only 24% of the overall time for n = 400, but almost 64% for n = 2000.

For the values of n that we were interested in for the summary task ($n = 400$), the decoder is actually slower than the encoder. This is due to the last linear layer which projects the embeddings on the vocabulary space. The main purpose of this study was to find ways to speed up the encoder, so little time was spent to figure out how to optimize the decoder. One way to increase the speed of the decoder is to decrease the size of the output vocabulary by using a subword vocabulary such as Byte-Pair Encoding [13] . However, this would increase the number of input and output tokens. Hence, a trade-off has to be found between a small vocabulary size and a small sequence size.

For values of n smaller than 2000, components linear in the size of the input are the ones that takes most time. Hence, we should not be surprised to see linear curves when displaying the time spent per sentence by the transformer and models with similar architecture.

Components	Complexity	Time (s) with n=400	Time (s) with n=2000
Encoder	$n*d*N*(n+d')$	0.0147	0.1522
Input embedding + positional encoding	$n*d$	<10^{-4}	<10^{-4}
Each Add and norm	$n*d*N$	0.0005 (1.6%)	0.0054 (2.9%)
Multi-head self-attention	n^2*d*N	0.0075 (24.2%)	**0.1206 (63.8%)**
Feed-forward network	$n*d*d'*N$	0.0055 (17.74%)	0.028 (14.8%)
Decoder	$m*d*(n*N+V)$	0.0182	0.0359
Input embedding + positional encoding	$m*d$	<10^{-4}	<10^{-4}
Each Add and norm	$m*d*N$	<10^{-4}	<10^{-4}
Multi-head self-attention	m^2*d*N	0.0011 (3.6%)	0.0023 (1.2%)
Multi-head src-to-target attention	$m*n*d*N$	0.0029 (9.4%)	0.0123 (6.5%)
Feed-forward network	$m*d*d'*N$	0.002 (6.5%)	0.0007 (0.4%)
Projection on vocabulary	$m*d*V$	**0.0115 (37.1%)**	0.0197 (10.4%)

Table 2: Complexity and running speed of the different components of Transformer

Hence, the quadratic complexity of the transformer only becomes the major issue when input sizes gets to at least 1500-2000. Even though it doesn't concern datasets such as CNN/DailyMail, it is still something useful to have in mind as some other datasets, such as the ones used to generate Wikipedia [9], exceed this size.

6 Conclusion

We have shown that the Transformer architecture can successfully be used for Text Summarization. Even though our project aimed at reducing the quadratic cost of self-attention for long-sequences, it turns out that this cost is not the main speed factor for articles of reasonable lengths (400-1000 words) and we have provided a detailed analysis of the complexity and runtime of all the Transformer components. We have shown that faster variants of the Transformer can be used for Text Summarization without losing too much abstractive power. Our main finding is that models that focus on extracting information at a local level outperform the Transformer. In that regard, the lightweight convolutions model of [8] and our local transformer model are most suited to Text Summarization.

The next steps of this project would be to produce the same analysis for larger architectures, trained until convergence. In order to improve our ROUGE score performances we would add Text Summarization specific components to our model such as a copy mechanism and a coverage mechanism.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[2] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701, 2015.

[5] Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. Pointing the unknown words. *arXiv preprint arXiv:1603.08148*, 2016.

[6] Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*, 2016.

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

[8] Felix Wu, Angela Fan, Alexei Baevski, Yann N Dauphin, and Michael Auli. Pay less attention with lightweight and dynamic convolutions. *arXiv preprint arXiv:1901.10430*, 2019.

[9] Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

[10] Myle Ott Sergey Edunov and Sam Gross. fairseq. `https://github.com/pytorch/fairseq`, 2019.

[11] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*, 2004.

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.