Long Document Summarization

Ziming Mao

28/9/2021

Introduction
- What is Summarization
- Abstractive vs. Extractive
- Transformers
- Challenges introduced by Long Documents

Datasets
- Paper Summarization
- Dialogue Summarization
- Other types of Summarization (Report, Screenplay, Book, Patent)

Prior Work
- Input Truncation
- Sparse Attention
- Extract-Abstract
- Divide and Conquer
- Hierarchical Models

First Paper
- Longformer
- Sparse encoder self attention
  - Sliding Window Local Attention
  - Global Attention

Second Paper
- Efficient Attention for Long Document Summarization
- Encoder-decoder Attention
- GovReport Dataset
- QA-based evaluation metric (APES_src)

Evaluation (might skip)
- **Rouge**
- BertScore
- MoverScore
- Meteor
- Human Evaluation

Supplementary Paper
- BigBird
  - + Random Token Attention
- DANCER
  - Divide and Conquer
- Extract then Abstract
  - RL-based Method

Reformer (Backup Slides *)
- LSH Attention
- Chunked Feed Forward Layer
- Reversible Residual Layer
- Axial Positional Encoding
- Google Colab Demo

Conclusion

Future Directions

Discussion Questions

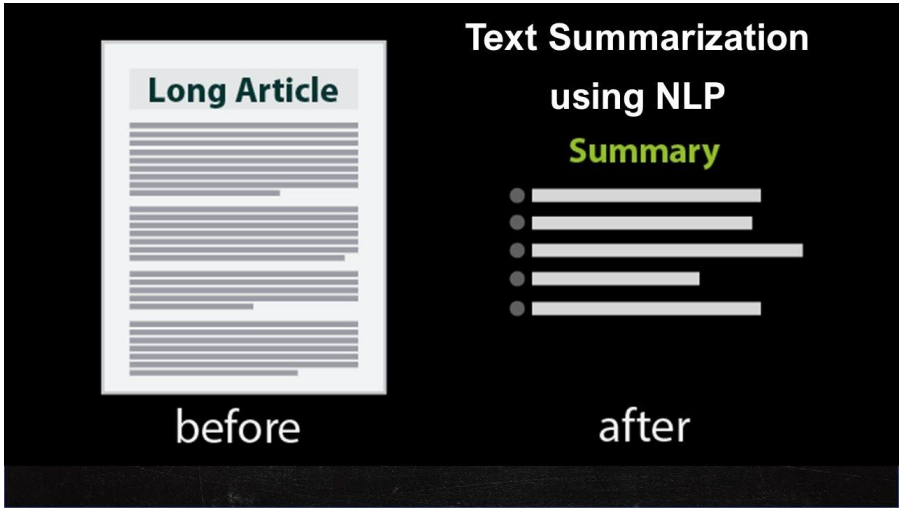
Introduction

- Summarization
- Abstractive vs. Extractive
- Transformers
- Challenges introduced by long document

Introduction

Summarization: Text summarization refers to the technique of shortening long pieces of text. The intention is to create a coherent and fluent summary having only the main points outlined in the document.

TL;DR **Condense a text into its main points!**

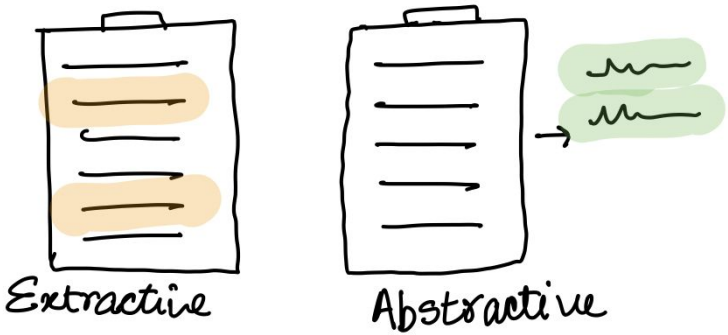


Credit: link

Introduction

Extractive vs. **Abstractive** Summarization

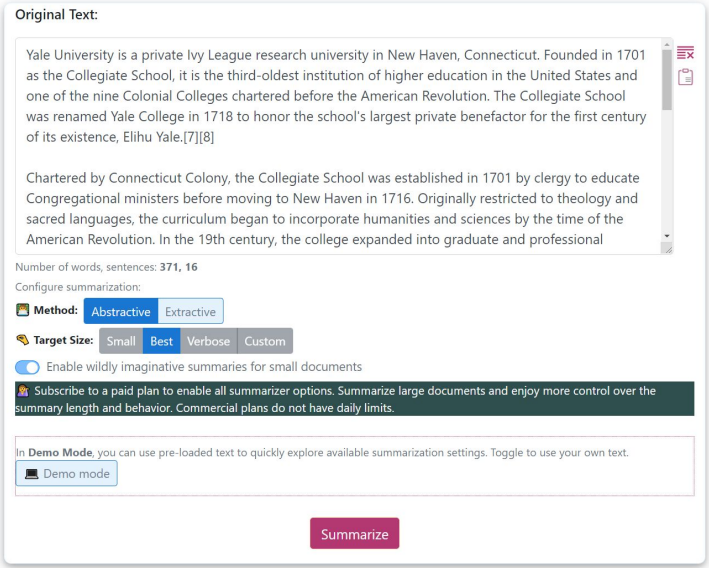
Extractive summarization is the strategy of concatenating extracts taken from a corpus into a summary, while **abstractive summarization** involves paraphrasing the corpus using novel sentences.

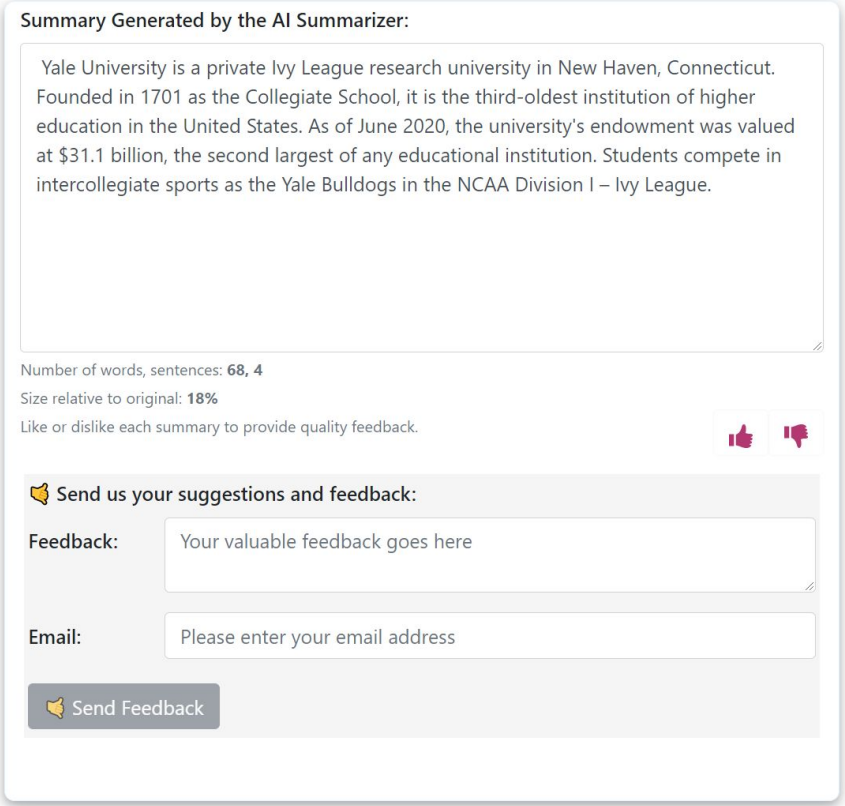


Credit: Extractive Text Summary using TextRank link

Introduction

- https://sassbook.com/ai-summarizer





Introduction

- **Transformers** have achieved SOTA on many summarization tasks.
- Long document cannot be easily tackled by Transformer models
  - **Memory cost** scales quadratically with input length
  - **How to keep as much information as possible given the memory constraint**

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

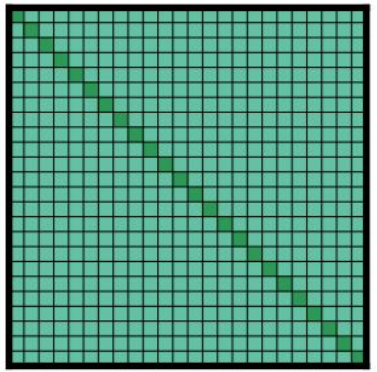
Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Abstract

Credit: Attention is All you Need paper

Introduction

- Transformers which uses attention to compute **pairwise relations between tokens**.
- Such framework has quadratic time and memory complexity, and is too costly for long documents.



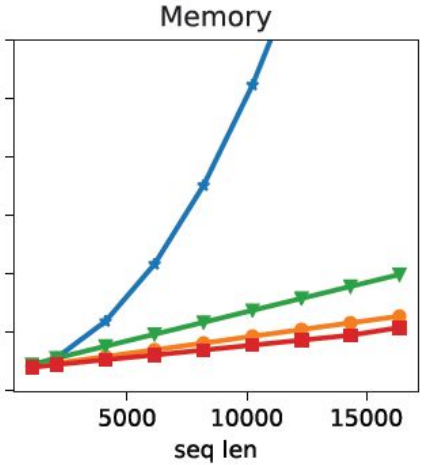
(a) Full n^2 attention

Credit: Longformer paper

Introduction

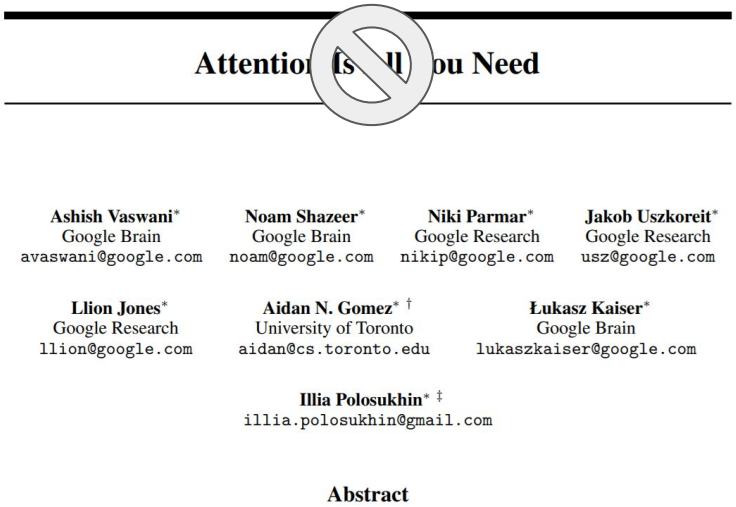
Quotation from the Longformer paper

A full attention model, if the input is 10k tokens, we end up with a cost of 70Gb of encoder self-attention and 10Gb of encoder-decoder attention



Blue - Full Attention
Other colors - Variants of Longformer
Credit: Longformer paper

Introduction



Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

Too much attention is not good either!

Datasets

- Paper Summarization
- Meeting Summarization
- Book Summarization
- Screenplay Summarization
- Report Summarization
- Patent Summarization

(and a lot more!)

Datasets

- **ArXiv** and **PubMed**
  - Scientific Papers + Abstract
- Can be easily downloaded from Hugging Face
- Three fields:
  - Article
  - Abstract
  - Paper title

name	train	validation	test
arxiv	203037	6436	6440
pubmed	119924	6633	6658

Dataset	# Doc	Summary		Doc	Comp.	Den.
		# word	# sent	# word		
PUBMED	133,215	202.4	6.8	3049.0	16.2	5.8
ARXIV	215,913	272.7	9.6	6029.9	39.8	3.8
BILLSUM	23,455	207.7	7.2	1813.0	13.6	4.1
BIGPATENT	1,341,362	116.5	3.7	3573.2	36.3	2.4
GOVREPORT	19,466	**553.4**	**17.8**	**9409.4**	19.0	7.3

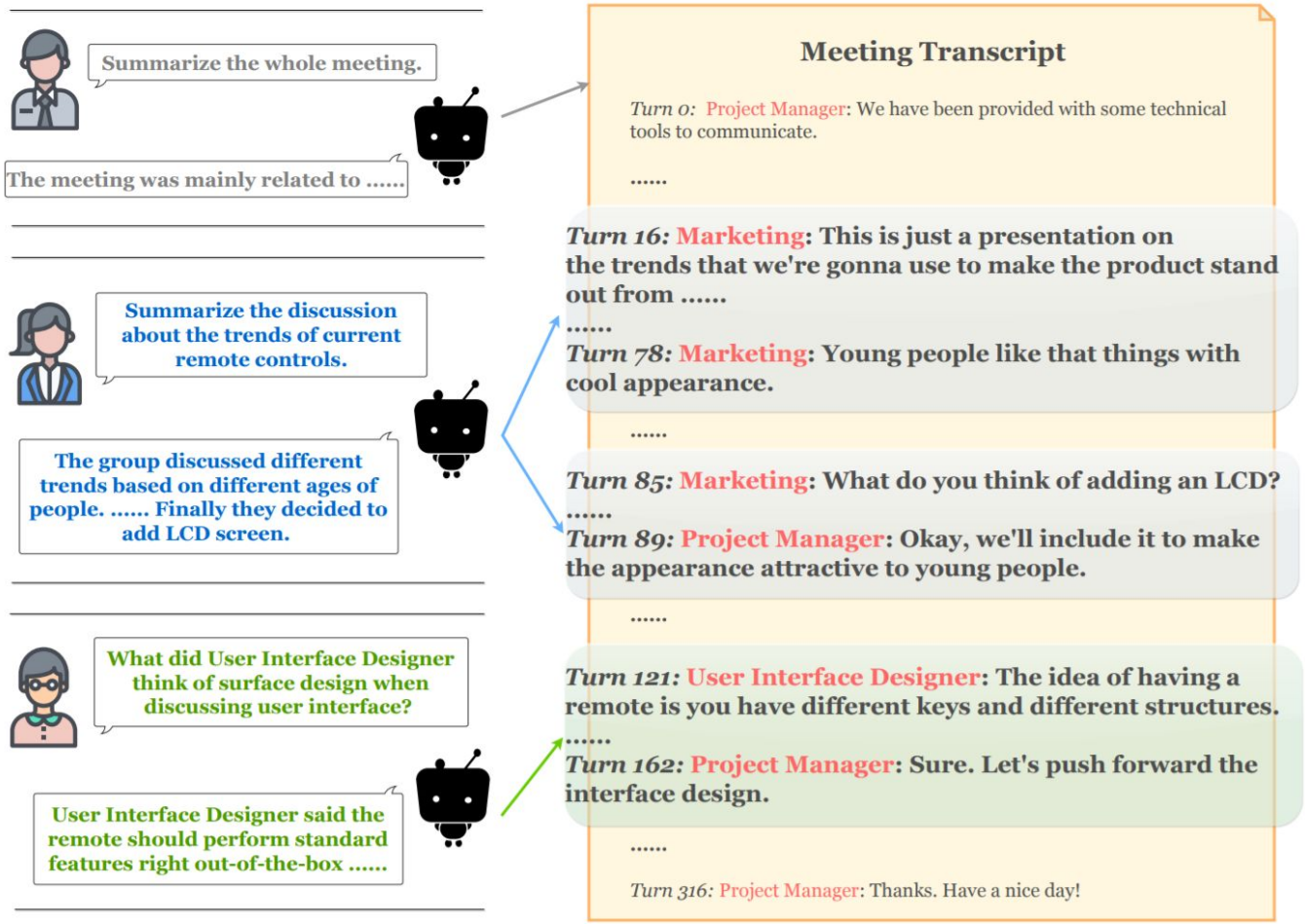
Long document summarization datasets comparison (Huang et al. 2021) link

Datasets

- Meeting Summarization Datasets
  - **AMI**, **ICSI**
    - Old dataset
  - **QMSum**
    - Query-based

ID	Dataset	# instances	# tokens (input)	# tokens (summary)	# speakers	Abstractive	Extractive	Domain
1	AMI	137	4757.0	322.0	4.0	√	√	Meetings
2	ICSI	59	10189.0	534.0	6.2	√	√	Meetings
3	SAMSum	16.4k	83.9	20.3	2.2	√		ChitChat
4	MediaSum	463.6k	1553.7	14.4	6.5	√		News Interviews
5	QMSum	1.8k	9069.8	69.6	9.2	√		Meetings
6	SUMMSCREEN	26.9k	6612.5	337.4	28.3	√		Television Series
7	SumTitles	21.4k	423.06	55.03	4.88	√		Movie
8	DialogSum	13.4k	131	13.8	-	√		Spoken
9	GupShup	16.4k	**83.9**	20.3	2.2	√		Cross-lingual
10	LCSPIRT	38500	684.3	75	2	√		Police

New Development in Dialogue Summarization (Feng et al. 2021)



QMSum Paper [link]

Introduction

- Book Summarization
  - **BookSum**
    - Paragraph
    - Chapter
    - Full book
- Screenplay summarization
  - **SummScreen**
- Patent Document Summarization
  - **BigPatent**
- Scientific Papers Summarization
  - **Scisumnet**
- **GovReport** - we will take about it later!

Dataset	# Docs.	Coverage	Density	Comp. Ratio	# Tokens	
					Source	Summary
Arxiv/PubMed	346,187	0.87	3.94	31.17	5179.22	257.44
BigPatent	1,341,306	0.86	2.38	36.84	3629.04	116.66
CNN/DM	311,971	0.85	3.47	14.89	803.67	59.72
Newsroom	1,212,739	0.83	9.51	43.64	799.32	31.18
XSum	226,677	0.66	1.09	19.25	438.43	23.89
NovelChapters*	8,088	-	-	-	5,165	372
BookSum Paragraph (ours)	142,753	0.50	0.92	6.47	159.55	40.59
BookSum Chapter (ours)	12,293	0.78	1.69	15.97	5101.88	505.42
BookSum Full (ours)	436	0.89	1.83	126.22	112885.15	1167.20

Table 2: Statistics of the BookSum data collection compared with other popular text summarization datasets. *NovelChapters dataset (Ladhak et al., 2020) could not be reliably reproduced at the time of writing of this work, the numbers were copied from the original paper.

BookSum (Kryściński et al. 2021)

Transcript:

[The apartment]

Sheldon : What color would you like to be ?
Leonard : Well , I 'd like to be green , but you know you always take it .
Sheldon : That 's not true . Any color 's fine with me . Yeah , I could be a - a combination of blue and yellow .
Leonard : Blue and yellow make green .
Sheldon : Well , then it 's settled .

Penny : Hi . Ready to go ?
Sheldon : Oh , good news , we ordered lunch , so we can all stay here and play Lord of the Rings Risk .
Amy : Sheldon , we said that we would play games with you tonight .
Sheldon : Oh , no , we 'll still be playing it tonight , this game can easily take eight hours .

Penny : Sweetie , you really thought I 'd want to do this ?
Leonard : No .
Penny : Well , did you tell him that ?
Leonard : Yes .
Penny : Did you say it out loud with words ?
Leonard : No .
Penny : I do n't want to spend the whole day playing a board game .

…

Recap:

Sheldon and Leonard are happy playing a board game until Amy and Penny say they are tired of doing what the guys want …

Figure 1: Excerpts from an example from SUMMSCREEN. The transcript and recap are from the TV show "The Big Bang Theory". Generating this sentence in the recap requires discerning the characters' feelings (clues in the transcript are underlined) about playing the board game (references are shown in red). Colored boxes indicate utterances belonging to the same conversations.

SummScreen dataset

Sample CNN/Daily Mail News Summary

An *explosion* rocks a chemical plant in China's south-eastern Fujian province for the second time in two years. Six were injured after the *explosion* and are being hospitalized. The *explosion* was triggered by an oil leak, though local media has not reported any toxic chemical spills.

Sample BIGPATENT Summary

A *shoelace cover* incorporating an interchangeable fashion panel for covering the *shoelaces* of a *gym shoe*. The *shoelace cover* is secured to the shoe by a number of *straps* threaded through slots in the *shoelace cover*. These *straps* secured to each side of the *gym shoe* include a loop and hook material such that the *straps* can be disengaged and the *shoelace cover* can be drawn back to expose the *shoelaces*. . .

Figure 1: Sample summaries from CNN/Daily Mail and BIGPATENT. Extractive fragments reused from input are underlined. Repeated entities indicating discourse structure are highlighted in respective colors.

BigPatent dataset

Prior Work

- Input Truncation
- Sparse Attention Mechanism
- Extract-Abstract Method
- Divide and Conquer
- Hierarchical Models

(and other task specific Methods)

Prior Work

Common approaches that deal with long input summarization

- **Truncating Input to a fixed length**
  - Assume that more information is located at the earlier part of the text
- **Sparse Attention**
  - Modify attention mechanism such that handling longer input isn't that costly
- **Extract than Abstract Method**
- **Task specific optimizations**
  - Such as Divide and Conquer
- **Hierarchical models**
  - HMNet & HAT-Bart

- Ask the professor to buy more GPUs

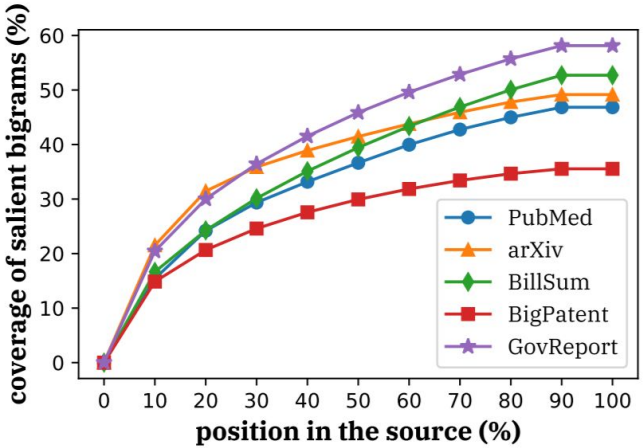
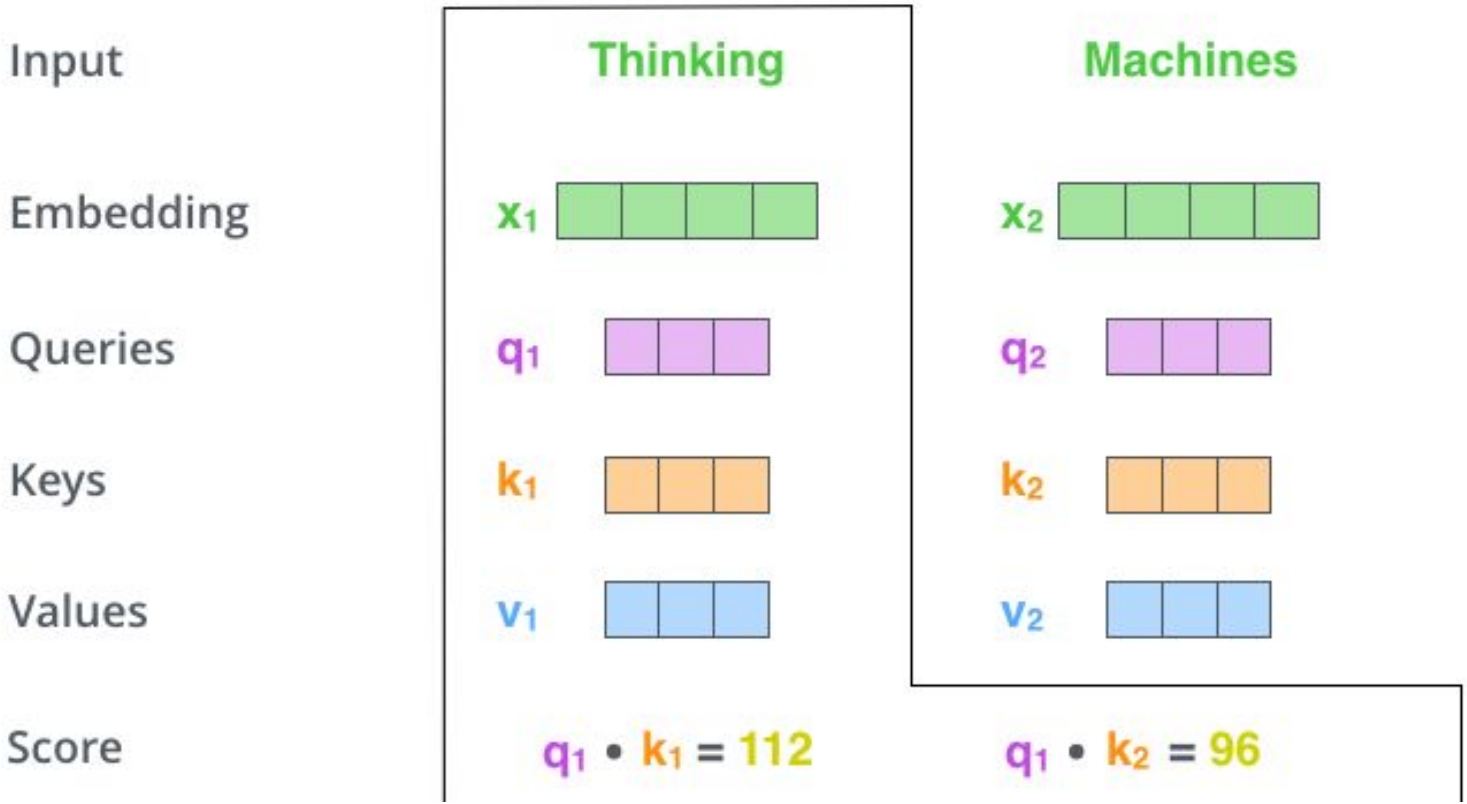


Figure 2: Percentage of unique salient bigrams accumulated from the start to X% of the source. Key information is spread over the documents in GovReport, highlighting the importance of understanding longer text.

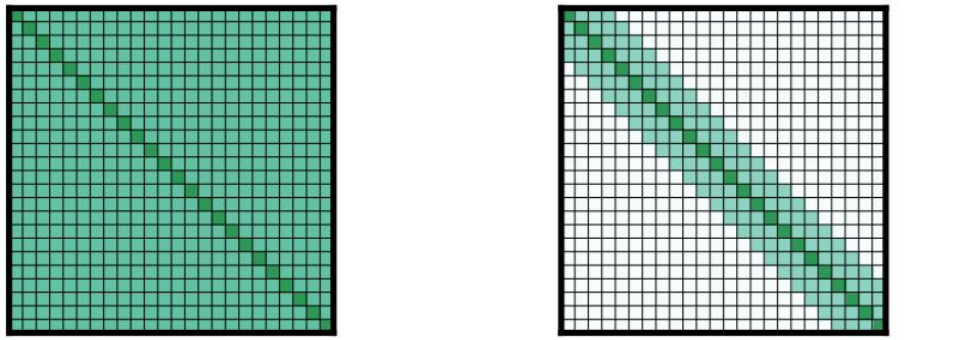
Credit: GovReport Paper


Input
Thinking
Machines
Embedding
x1
x2
Queries
q1
q2
Keys
k1
k2
Values
v1
v2
Score
q1 • k1 = 112
q1 • k2 = 96

Prior Work

Sparse Attention Mechanism

Sparse attention reduces computation time and the memory requirements of the attention mechanism by **computing a limited selection of similarity scores from a sequence rather than all possible pairs**, resulting in a sparse matrix rather than a full matrix



(a) Full n^2 attention (b) Sliding window attention

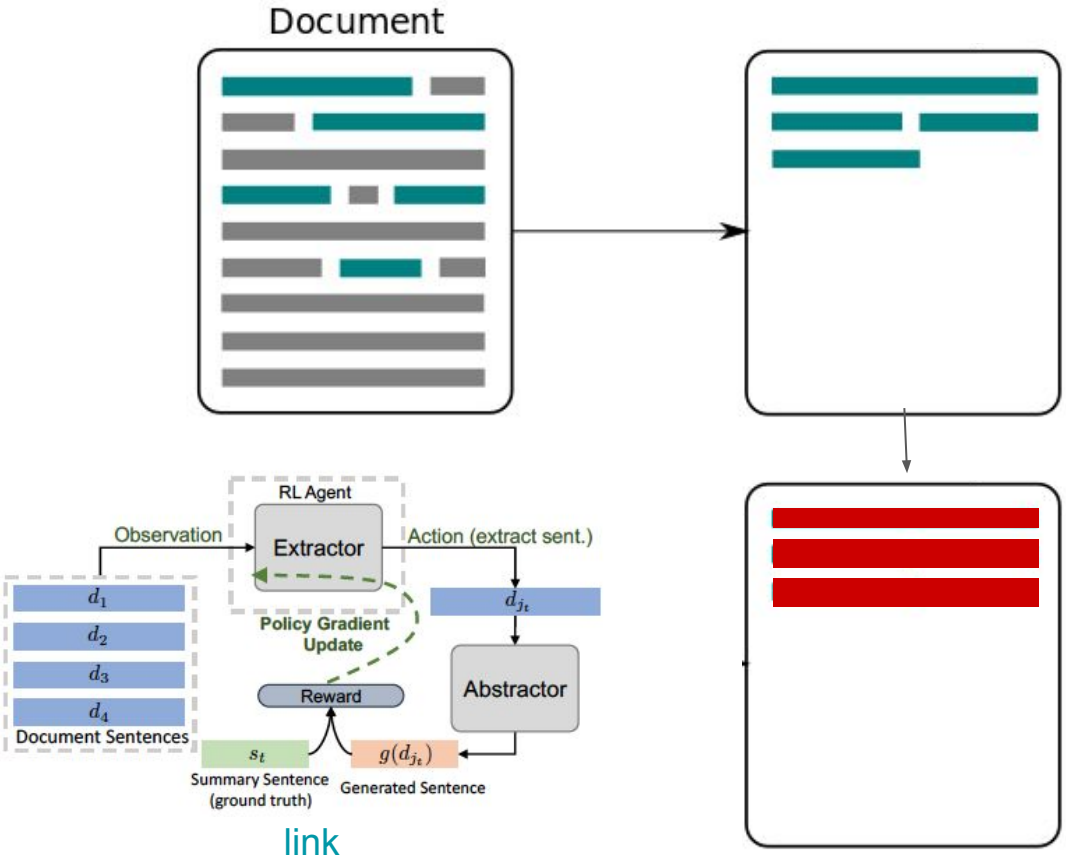
Credit: Longformer paper

Earlier Work

Extract than abstract Model

- Train an extractor that picks out salient text spans
- Use another summarize to summarize these text spans

- Challenge: how to reduce information loss?
  - Naive: Completely separate
  - RL? (Chen, 2018)



[link]

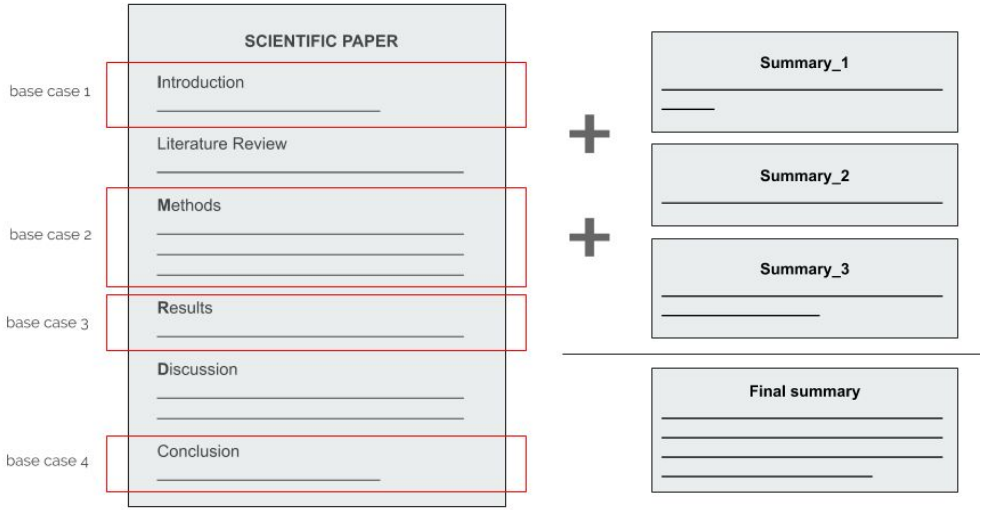
Earlier Work

Task specific Methods

One example: Scientific Documents

Divide and Conquer!

- **Divide** (how?)
- **Extract + Summarize**
- **Summarize + Summarize**
- **etc.**

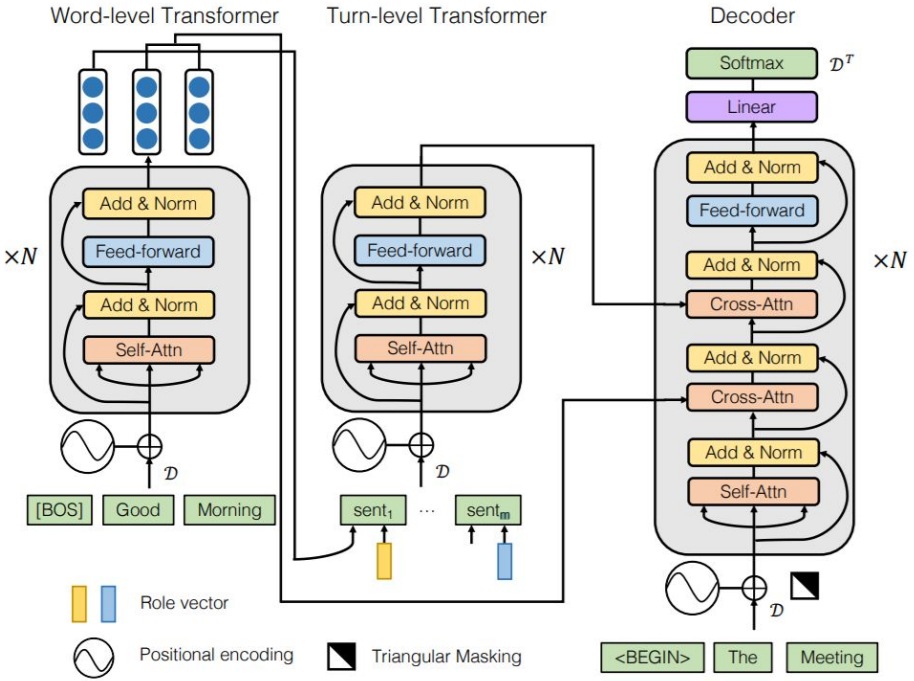


Credit: Divide and Conquer approach on summarizing scientific papers [link]

Earlier Work

Hierarchical Models

- HMNet
  - Encode each token in one turn using a trainable embedding matrix
  - Also train **two embedding matrices to represent the part-of-speech (POS) and entity (ENT) tags**
  - Vector to **represent speaker roles** and concat to turn input
  - Decoder transformer block includes two cross attention layers: it first looks at token-level outputs, then turn-level outputs

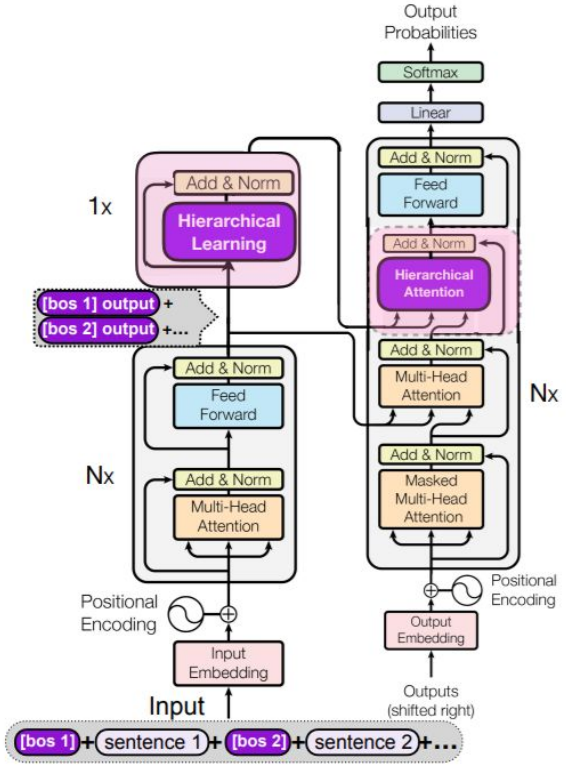


Credit: HMNet Paper link

Earlier Work

Hierarchical Models

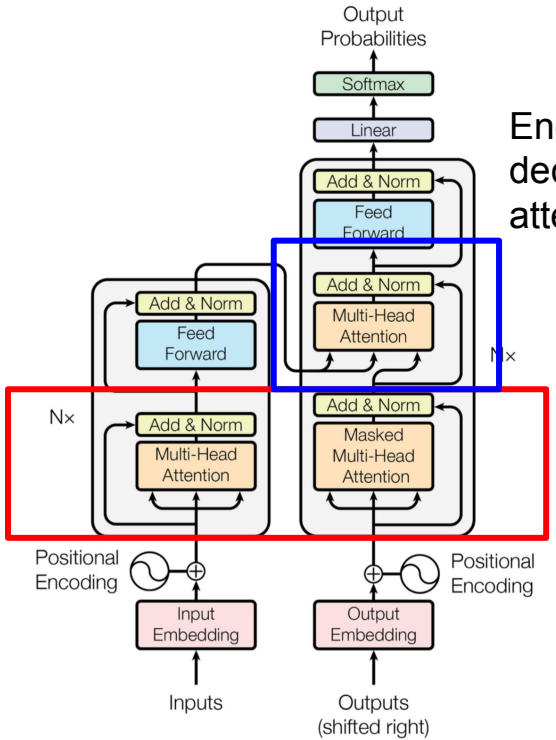
- HAT-BART
  - Hierarchical attention attends to all [bos] tokens
  - “We refer to this layer as the hierarchical encoder layer, which produces another level of **contextual representations** for each of the BOS tokens, which can be interpreted as sentence level representations. “
  - Decoder: “We add an attention module that attends over the BOS token embeddings from the hierarchical encoder layer”



Credit: HAT-BART link

Why these two papers

- Longformer - is a solution to reduce encoder self-attention
- Second paper tries to reduce the cost of **encoder-decoder attentions in summarization models**
  - Dataset: GovReport
  - New QA-based metric



Encoder-decoder attention

Two Self-attention (encoder + decoder)

Figure 1: The Transformer - model architecture.

First Paper

Longformer: The Long-Document Transformer

Dec. 2020

Beltagy, et. al.

Paper 1 - Longformer

- Longformer: The Long-Document Transformer, 2020 [link]

Longformer: The Long-Document Transformer

Iz Beltagy* **Matthew E. Peters*** **Arman Cohan***
Allen Institute for Artificial Intelligence, Seattle, WA, USA
{beltagy,matthewp,armanc}@allenai.org

Abstract

Transformer-based models are unable to process long sequences due to their self-attention operation, which scales quadratically with the sequence length. To address this limitation, we introduce the Longformer with an attention mechanism that scales linearly with sequence length, making it easy to process documents of thousands of tokens or longer. Longformer's attention mechanism is a drop-in replacement for the standard self-attention and combines a local windowed attention with a task motivated global attention. Following prior work on long-sequence transformers, we evaluate Longformer on character-level language modeling and achieve state-of-the-art results on `text8` and `enwik8`. In contrast to most prior work, we also pretrain Longformer and finetune it on a variety of downstream tasks. Our pretrained Longformer consistently outperforms RoBERTa on long document tasks and sets new state-of-the-art results on WikiHop and TriviaQA. We finally introduce the Longformer-Encoder-Decoder (LED), a Longformer variant for supporting long document generative sequence-to-sequence tasks, and demonstrate its effectiveness on the arXiv summarization dataset.[1]

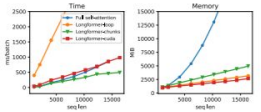


Figure 1: Runtime and memory of full self-attention and different implementations of Longformer's self-attention; `Longformer-loop` is non-vectorized, `Longformer-chunk` is vectorized, and `Longformer-cuda` is a custom cuda kernel implementations. Longformer's memory usage scales linearly with the sequence length, unlike the full self-attention mechanism that runs out of memory for long sequences on current GPUs. Different implementations vary in speed, with the vectorized `Longformer-chunk` being the fastest. More details are in section 3.2.

1 Introduction

Transformers (Vaswani et al., 2017) have achieved state-of-the-art results in a wide range of natural language tasks including generative language modeling (Dai et al., 2019; Radford et al., 2019) and discriminative language understanding (Devlin et al., 2019). This success is partly due to the self-attention component which enables the network to capture contextual information from the entire sequence. While powerful, the memory and computational requirements of self-attention grow quadratically with sequence length, making it infeasible (or very expensive) to process long sequences.

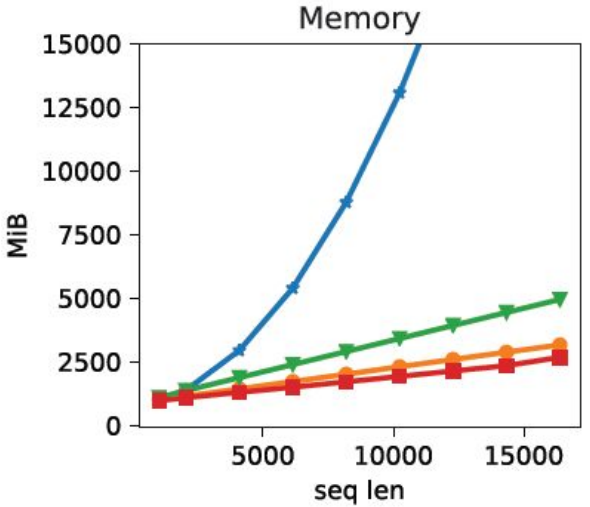
To address this limitation, we present Longformer, a modified Transformer architecture with a self-attention operation that scales linearly with the sequence length, making it versatile for processing long documents (Fig 1). This is an advantage for natural language tasks such as long document classification, question answering (QA), and coreference resolution, where existing approaches partition or shorten the long context into smaller sequences that fall within the typical 512 token limit of BERT-style pretrained models. Such partitioning could potentially result in loss of important cross-partition information, and to mitigate this problem, existing methods often rely on complex architectures to address such interactions. On the other hand, our proposed Longformer is able to build contextual representations of the entire context using multiple layers of attention, reducing the

* Equal contribution.
[1] https://github.com/allenai/longformer

Paper 1 - Longformer

- Problem Statement:
  - “the memory and computational requirements of self-attention grow quadratically with sequence length, making it infeasible (or very expensive) to process long sequences.”

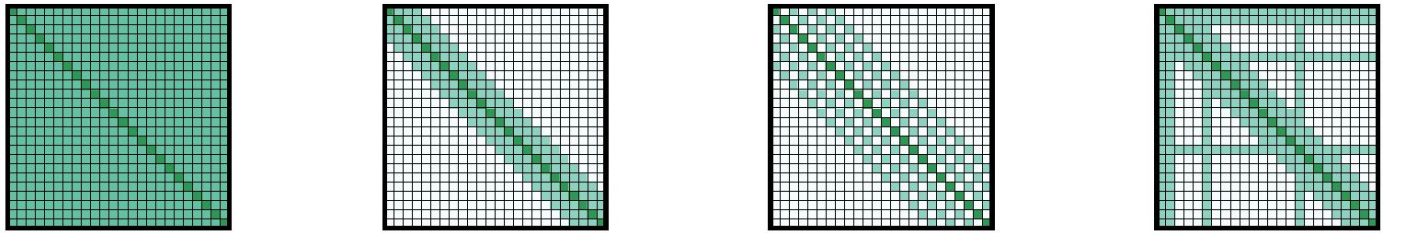
- A full attention model, if the input is **10k tokens**, we end up with a cost of **70Gb** of encoder self-attention and **10Gb** of encoder-decoder attention
- BERT, for example, sets max seq length to be **512/1024 tokens**



Blue - Full Attention
Other colors - Variants of Longformer

Paper 1 - Longformer

- Sparse Attention Mechanism
  - “The original Transformer model has a self-attention component with O(n2) time and memory complexity where n is the input sequence length. To address this challenge, we sparsify the full self-attention matrix according to an **“attention pattern” specifying pairs of input locations attending to one another**. Unlike the full self-attention, our proposed attention pattern scales linearly with the input sequence, making it efficient for longer sequences.”



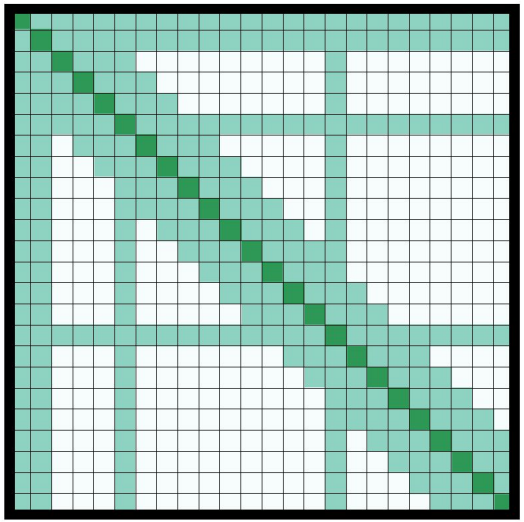
(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer.

Paper 1 - Longformer

- **Global Attention**
  - For example, for classification, global attention is used for the [CLS] token while in QA global attention is provided on all question tokens

- **Sliding Window Attention**
  - “Given the importance of local context, our attention pattern employs a fixed-size window attention surrounding each token.”



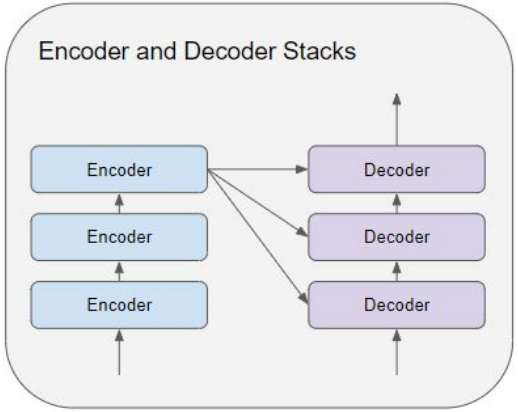
(d) Global+sliding window

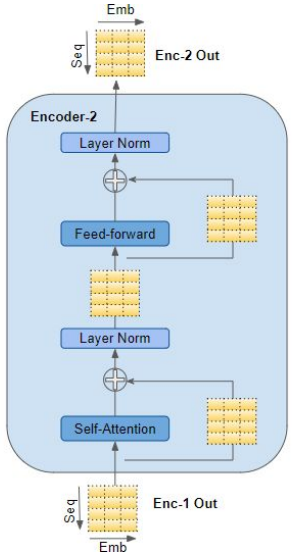
Paper 1 - Longformer

- “On the other hand, our proposed Longformer is able to build contextual representations of the entire context using **multiple layers of attention**.”

- “We **use small window sizes for the lower layers and increase window sizes as we move to higher layers.** This allows the top layers to learn higher-level representation of the entire sequence while having the lower layers capture local information.”

- **Stacking many layers of such sliding windows on top of each other** (which is common with any Transformer) still leads to a big receptive field.





Credit: https://jalammar.github.io/illustrated-transformer/

Paper 1 - Longformer

	QA			Coref.	Classification	
Model	WikiHop	TriviaQA	HotpotQA	OntoNotes	IMDB	Hyperpartisan
RoBERTa-base	72.4	74.3	63.5	78.4	95.3	87.4
Longformer-base	**75.0**	**75.2**	**64.4**	**78.6**	**95.7**	**94.8**

Table 7: Summary of finetuning results on QA, coreference resolution, and document classification. Results are on the development sets comparing our Longformer-base with RoBERTa-base. TriviaQA, Hyperpartisan metrics are F1, WikiHop and IMDB use accuracy, HotpotQA is joint F1, OntoNotes is average F1.

Longformer was applied to a variety of long document tasks: question answering, coreference resolution, classification

Paper 1 - Longformer

- **LED** (Longformer-Encoder-Decoder)
- “To facilitate modeling long sequences for seq2seq learning, we propose a Longformer variant that has both the encoder and decoder Transformer stacks but instead of the full self-attention in the encoder, **it uses the efficient local + global attention pattern of the Longformer.**“
- “**The decoder uses the full self-attention to the entire encoded tokens and to previously decoded locations**. We call this model Longformer-Encoder-Decoder (LED) which scales linearly with the input.”
- LED extends positional embedding to **16k tokens**

Paper 1 - Longformer

- Experiments on ArXiv dataset

- Slightly outperforming BigBird
- Author believes that **BigBird** benefited from better pretraining
  - BigBird starts from and continues pre-training Pegasus, **a model specifically pre-trained for summarization**
- The author believes that with better pretraining, there is still room for improving LED

	R-1	R-2	R-L
Discourse-aware (2018)	35.80	11.05	31.80
Extr-Abst-TLM (2020)	41.62	14.69	38.03
Dancer (2020)	42.70	16.54	38.44
Pegasus (2020)	44.21	16.95	38.83
LED-large (seqlen: 4,096) (ours)	44.40	17.94	39.76
BigBird (seqlen: 4,096) (2020)	**46.63**	19.02	41.77
LED-large (seqlen: 16,384) (ours)	**46.63**	**19.62**	**41.83**

Table 11: Summarization results of Longformer-Encoder-Decoder (LED) on the arXiv dataset. Metrics from left to right are ROUGE-1, ROUGE-2 and ROUGE-L.

Second Paper

Efficient Attentions for Long Document Summarization

April. 2021

Huang, et. al.

Paper 2

- Efficient Attentions for Long Document Summarization (Second Paper), 2021 [link]

Efficient Attentions for Long Document Summarization

Luyang Huang [1] Shuyang Cao[1] Nikolaus Parulian[2] Heng Ji[2] Lu Wang[1]
[1]Computer Science and Engineering, University of Michigan, Ann Arbor, MI
[2]Department of Computer Science, University of Illinois at Urbana-Champaign, IL
[1]{lyhuang, caoshuy, wangluxy}@umich.edu
[2]{nnp2, hengji}@illinois.edu

Abstract

The quadratic computational and memory complexities of large Transformers have limited their scalability for long document summarization. In this paper, we propose **HEPOS**, a novel *efficient encoder-decoder attention with head-wise positional strides* to effectively pinpoint salient information from the source. We further conduct a systematic study of existing efficient self-attentions. Combined with HEPOS, we are able to process ten times more tokens than existing models that use full attentions. For evaluation, we present a new dataset, **GOVREPORT**, with significantly longer documents and summaries. Results show that our models produce significantly higher ROUGE scores than competitive comparisons, including new state-of-the-art results on PubMed. Human evaluation also shows that our models generate more informative summaries with fewer unfaithful errors.

1 Introduction

Long documents, such as scientific papers and government reports, often discuss substantial issues at length, and thus are time-consuming to read, let alone to comprehend. Generating abstractive summaries can help readers quickly grasp the main topics, yet prior work has mostly focused on short texts (containing hundreds of words), e.g., news articles (Gehrmann et al., 2018; Liu and Lapata, 2019; Zhang et al., 2019).

Model training efficiency and *summary quality* present a pair of challenges for long document summarization. State-of-the-art systems (Lewis et al., 2020; Zhang et al., 2019) are built upon Transformer (Vaswani et al., 2017), which uses attentions to compute pairwise relations between tokens. Such framework has quadratic time and memory complexities, and is too costly for long documents [1]. Solutions have been proposed to reduce the calculation of *encoder self-attentions* (Wang et al., 2020c; Zaheer et al., 2020) by selectively attending to neighboring tokens (Beltagy et al., 2020; Child et al., 2019) or relevant words (Kitaev et al., 2020; Tay et al., 2020a). Yet, these methods do not apply to *encoder-decoder attentions* in summarization models since they collaborate and dynamically pinpoint salient content in the source as the summary is decoded. Truncation is commonly used to circumvent the issue. However, training on curtailed content further aggravates "hallucination" in existing abstractive models (Maynez et al., 2020).

We argue that summarizing long documents (e.g., with thousands of words or more) requires efficient handling of both types of attentions. To this end, we propose an efficient encoder-decoder attention with **head-wise positional strides** (**HEPOS**), where the attention heads follow a strided pattern and have varying starting positions. HEPOS reduces computational and memory costs while (1) maintaining the power of emphasizing important tokens, and (2) preserving the global context per head. HEPOS successfully doubles the processed input sequence size, when combined with any encoder. To the best of our knowledge, we are the first to study efficient encoder-decoder attentions and provide a systematic comparison of diverse encoder attentions for the task of summarization.[2]

For evaluation, we collect **a new large-scale dataset, GOVREPORT**, consisting of about 19.5k U.S. government reports with expert-written abstractive summaries.[3] GOVREPORT has two important features: (1) It contains significantly longer documents (9.4k words) and summaries (553 words) than existing datasets, such as PubMed and arXiv (Cohan et al., 2018) (see Table 2); (2) Salient

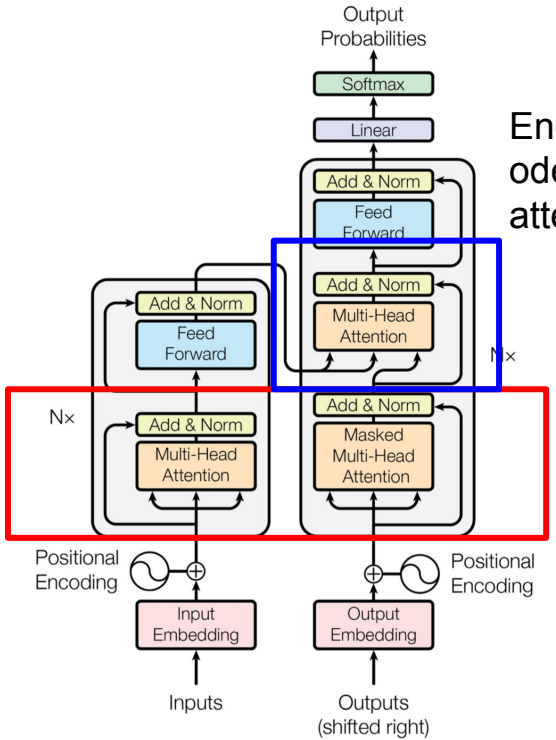
[1]For instance, to fine-tune BART on documents of 10K tokens with a batch size of 1, 70GB of memory is needed for encoder attentions, and 8GB for encoder-decoder attentions.

[2]Our code is released at https://github.com/luyang-huang96/LongDocSum.

[3]GOVREPORT can be downloaded from https://gov-report-data.github.io.

Paper 2

- Longformer - is a solution to reduce the memory cost of **encoder self-attention**
- This paper tries to reduce the cost of **encoder-decoder attentions in summarization models**
- This paper proposes “an efficient encoder-decoder attention with **head-wise positional strides (HEPOS)**”.



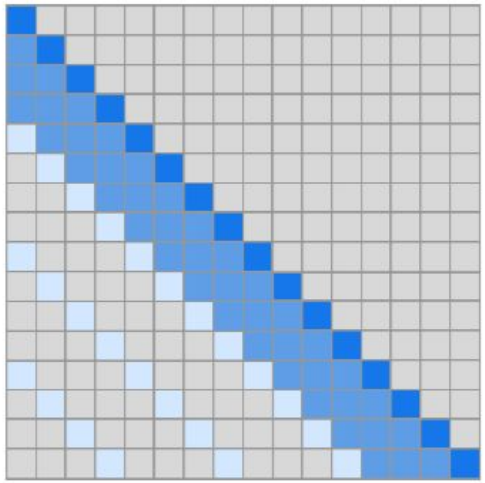
Encoder-decoder attention

Encoder self-attention and decoder self-attention

Figure 1: The Transformer - model architecture.

Paper 2

- Sparse Attentions:
  - Sliding Window Attentions
  - Adaptive Span (learn window size at different layers)
  - **Stride Patterns (capture long term interactions, each query attends to every s-th token)**
  - Learnable sparse patterns
    - *LSH (locality-sensitive hashing) attentions (*will go through if have time: Reformer paper)
    - Sinkhorn attentions

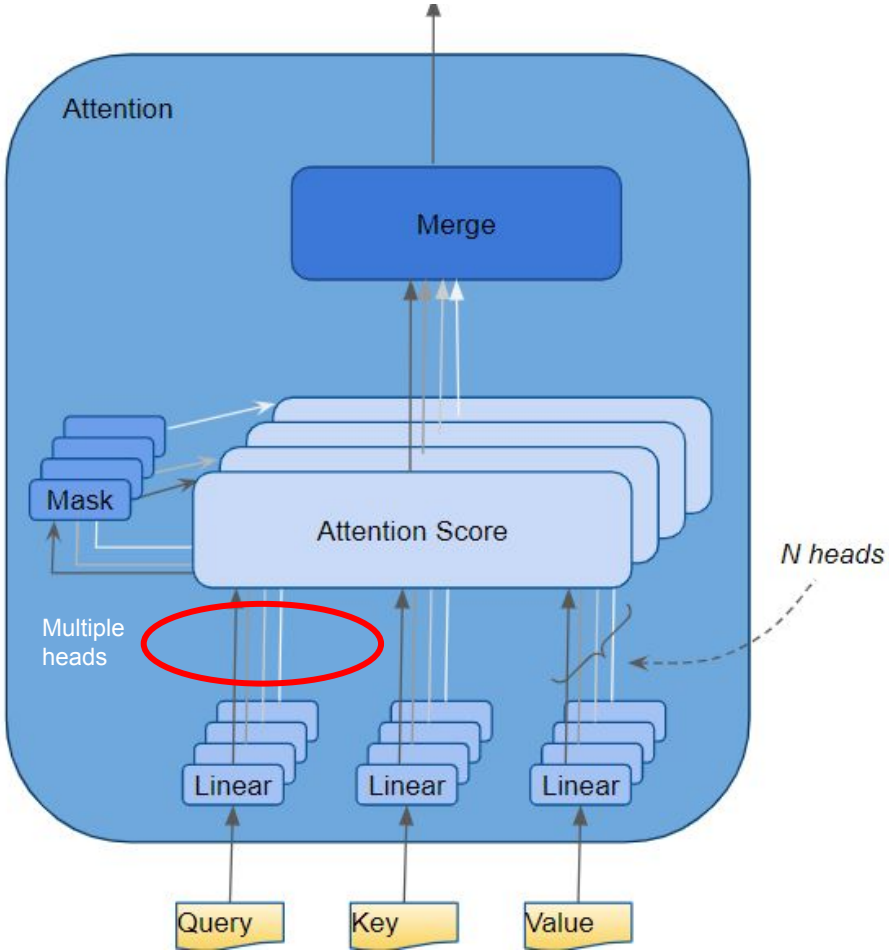


(b) Sparse Transformer (strided)

link

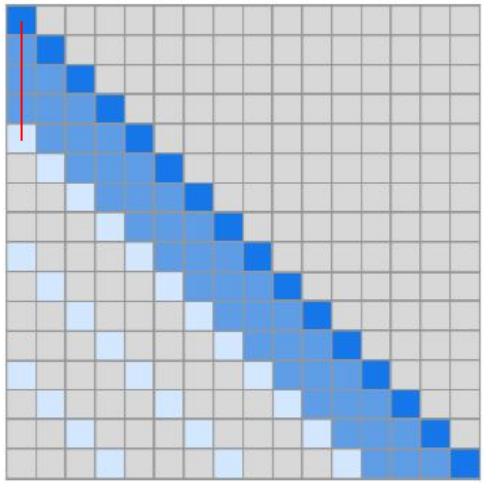
Paper 2

- Recall…



Paper 2

- Stride Patterns in encoder self-attention
- One Head attend to the previous **L** locations, and the other head attends to the every **L**-th locations, where **L** is the stride.



(b) Sparse Transformer (strided)

link

Paper 2

- Observations:
  - Attention heads are redundant and any individual head **rarely attends to several tokens in a row**
  - HEPOS uses separate encoder-decoder heads on the same layer to **cover different subsets of source tokens at fixed intervals**

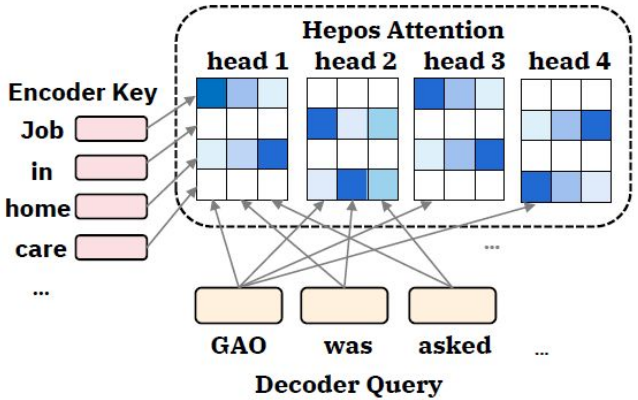


Figure 1: A toy example of our HEPOS attention, with a stride of 2 and four attention heads. Dark colors indicate that heads 1 and 3 attend to the first and third tokens ("Job" and "home") in the input, heads 2 and 4 look at the second and fourth words ("in" and "care").

Credit: https://arxiv.org/pdf/2104.02112.pdf

Paper 2

- HEPOS uses separate encoder- decoder heads on the same layer to cover different subsets of source tokens at fixed intervals.

- **Each head starts at a different position, and all heads collectively attend to the full sequence.**

Given a stride size of s_h, for the h-th head, its attention value between decoder query $\mathbf{q}_j$ (at step j) and encoder key vector $\mathbf{k}_i$ (for the i-th input token) can be formulated as:

$$a_{ji}^h = \begin{cases} \text{softmax}(\mathbf{q}_j \mathbf{k}_i), & \text{if } (i-h) \bmod s_h = 0 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

In HEPOS attention, each query token attends to n/s_h tokens per head, yielding a memory complexity of $\mathcal{O}(mn/s_h)$, where m is the output length.

Paper 2

System (MAXLEN)	GovReport R-1	GovReport R-2	GovReport R-L	PubMed R-1	PubMed R-2	PubMed R-L
Baselines						
PEGASUS (1024)	–	–	–	45.97	20.15	41.34
TLM (full)	–	–	–	42.13	16.27	39.21
SEAL (full)	–	–	–	46.50	20.10	42.20
DANCER (full)	–	–	–	46.34	19.97	42.42
BIGBIRD (3072)	–	–	–	46.32	20.65	42.33
Encoder variants w/ full enc-dec attn.						
FULL (1024)	52.83	20.50	50.14	45.36	18.74	40.26
STRIDE (4096)	54.29	20.80	51.35	46.95	19.98	41.67
LIN. (3072)	44.84	13.87	41.94	43.69	16.35	38.66
LSH (4096)	54.75	21.36	51.27	47.54	20.79	42.22
SINKHORN (5120)	55.45	21.45	52.48	47.96	20.78	42.53
Encoder variants w/ HEPOS enc-dec attn. (ours)						
LSH (7168)	55.00	21.13	51.67	**48.12**	**21.06**	**42.72**
SINKHORN (10240)	**56.86**	**22.62**	**53.82**	47.93	20.74	42.58

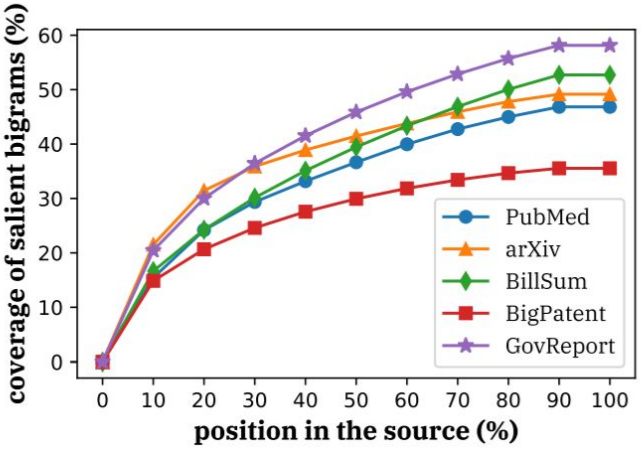
Table 4: ROUGE scores for models trained on the same GPU. SINKHORN with HEPOS enc-dec attention and LSH with HEPOS both read more text and obtain significantly better scores than other models on GovReport and PubMed ($p < 0.0005$).

Paper 2

- Proposed a new dataset **GovReport**
  - It contains significantly longer documents and summaries, then ArXiv and PubMed
  - **Salient content is spread throughout the documents, as opposed to concentrated at earlier parts of the document.**

Dataset	# Doc	Summary		Doc	Comp.	Den.
		# word	# sent	# word		
PUBMED	133,215	202.4	6.8	3049.0	16.2	5.8
ARXIV	215,913	272.7	9.6	6029.9	39.8	3.8
BILLSUM	23,455	207.7	7.2	1813.0	13.6	4.1
BIGPATENT	1,341,362	116.5	3.7	3573.2	36.3	2.4
GOVREPORT	19,466	**553.4**	**17.8**	**9409.4**	19.0	7.3

Table 2: Statistics of GOVREPORT and existing long document summarization datasets. **Comp.**: compression ratio, **Den.**: extractive fragment density (Grusky et al., 2018). All values are mean over the whole dataset except for the "# Doc" column. Documents and summaries in GOVREPORT are significantly longer.



Paper 2

- Query/aspect based summarization?

Notably, summaries of GAO reports are written by experts, and are often structured into three aspects in order: "`Why GAO did this study`"—motivation and problem(s) under discussion, "`What GAO found`"—findings of the report, and "`What GAO recommends`"—suggestions and solutions to the problem(s). All but

Paper 2

- Proposed a new evaluation metric for faithfulness
- With questions generated from references, the metric, APES_src, compares **QA answers by reading the source and the system summary.**

QA-based Evaluation. We present a new faithfulness evaluation metric by extending the **APES** score (Eyal et al., 2019). We follow APES to construct a set of **cloze questions**, $\{q\}$, from each reference summary by masking entities. Events, dates, and numbers are also masked, as they are prevalent in our data. Each masked phrase becomes the gold-standard answer a_{ref} for a question q. We do not

Paper 2

Original Reference Summary:

Arsenal beat Burnley 1-0 in the EPL. a goal from Aaron Ramsey secured all three points. win cuts Chelsea 's EPL lead to four points .

Produces questions:

Q: _____ beat @entity7 1-0 in the @entity4; **A**: Arsenal

Q: @entity0 beat _____ 1-0 in the @entity4; **A**: Burnley

Q: @entity0 beat @entity7 1-0 in the _____; **A**: EPL

Q: a goal from _____ secured all three points; **A**: Aaron Ramsey

Q: win cuts _____'s @entity4 lead to four points; **A**: Chelsea

Q: win cuts @entity19 's _____ lead to four points; **A**: EPL

Figure 3: Example 202 from the *CNN/Daily Mail* test set.

https://aclanthology.org/N19-1395/

QA-based Evaluation. We present a new faithfulness evaluation metric by extending the **APES** score (Eyal et al., 2019). We follow APES to construct a set of **cloze questions**, $\{q\}$, from each reference summary by masking entities. Events, dates, and numbers are also masked, as they are prevalent in our data. Each masked phrase becomes the gold-standard answer a_{ref} for a question q. We do not generate natural language questions (Durmus et al., 2020; Wang et al., 2020a), due to the lack of accurate question generation models for the domains of government reports and scientific papers.

Paper 2

- Context is obtained by greedy extraction based on Rouge-2

- Use QA model
  - **QA models are trained by reading a question and a text to label the answer span in the context.**
  - **a_{sys}** from predicted summary and question
  - **a_{ref}** from gold summary and question
  - **a_{cxt}** from context and question

QA models are trained by reading a question and a **context** to label the answer span in the context. We construct context by greedily selecting sentences that maximize the improvement of ROUGE-2 recall when compared with the reference summary. If the answer a_{ref} cannot be found in the context, the sample is excluded from training. We train all QA models by fine-tuning BERT (Devlin et al., 2019) to predict the answer span.

To evaluate the faithfulness of a system summary, **APES** uses the QA model to read the summary and a question q to label an answer a_{sys}. It calculates a unigram F1 score by *comparing* a_{sys} *and* a_{ref}. Different from APES, we further use the QA model to read the context (sentences selected from the source) and give an answer a_{cxt} to the question q. We compute a unigram F1 by *comparing* a_{sys} *and* a_{cxt}, denoted as $\textbf{APES}_{src}$. Given

Paper 2

6.4 Reading More Input Improves Faithfulness

Here we first show **human evaluation** results on informativeness and unfaithful errors in the generated summaries. We sample 100 documents from GovReport and PubMed (50 each) with structured references that are labeled with aspects as described in § 4 and Appendix D. Each sample is evaluated by two fluent English speakers, who have cumulatively annotated tens of thousands of sentences for the same tasks before this work. Annotators are asked to label each summary sentence with an aspect and then decide whether it contains any type of error. Three types of unfaithful errors are considered: (i) **hallucination**—fabricating content not present in the input, (ii) **deletion**—incorrectly

Paper 2

	GovReport		PubMed	
Metric	Inf.↑	Err.↓	Inf.↑	Err.↓
FactCC	0.07	-0.08	0.10	-0.14
APES	0.16	-0.15	0.25	-0.31
APES_{src}	**0.21**	**-0.23**∗	**0.32**∗	**-0.32**

Table 8: Pearson correlation between human ratings and metrics. We use aggregated unfaithful errors (Err.). ∗: significantly better than other metrics based on William's test (Williams, 1959) ($p < 0.05$).

QA-based metrics are better correlated with human ratings

But they are trickier to use!

Evaluation

- Rouge
- BertScore
- MoverScore
- Meteor

Evaluation

- For text summarization, there may exist two equally good summaries for a single document which focus on different content and are lexically diverse

Evaluation

- **ROUGE** performs evaluation by comparing the candidate summary to a set of human-produced reference summaries, specifically by computing the co-occurrences of **n-grams** between the candidate and each reference
- Commonly, people use **Rouge-1**, **Rouge-2**, and **Rouge-L**
- **Unigram, Bigram, Longest Common Subsequence**

$$\text{ROUGE-N} = \frac{\sum_{S \in \{References\}} \sum_{gram_n \in S} \text{Count}_{\text{match}}(gram_n)}{\sum_{S \in \{References\}} \sum_{gram_n \in S} \text{Count}(gram_n)} \quad (2.19)$$

Evaluation

- ROUGE … struggles to distinguish between good and bad summaries for the same source document (Bohm et al. ¨ [2019])

Reference:
https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1920-pg-projects/Technical-Writer-Assistant.pdf

Evaluation

- Human Evaluation
- Experiments are **costly** and **impractical**, especially on **longer and technical datasets such as arXiv and PubMed**. Furthermore, because of budget limitations, experiments are normally small-scale and compare to only a subset of the rival architectures.

Reference:
https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1920-pg-projects/Technical-Writer-Assistant.pdf

Evaluation

- BertScore
  - **BERTScore** uses **BERT** (Devlin et al. [2018]) to perform the automatic evaluation of two texts by comparing the weighted cosine similarities of their embedded representations

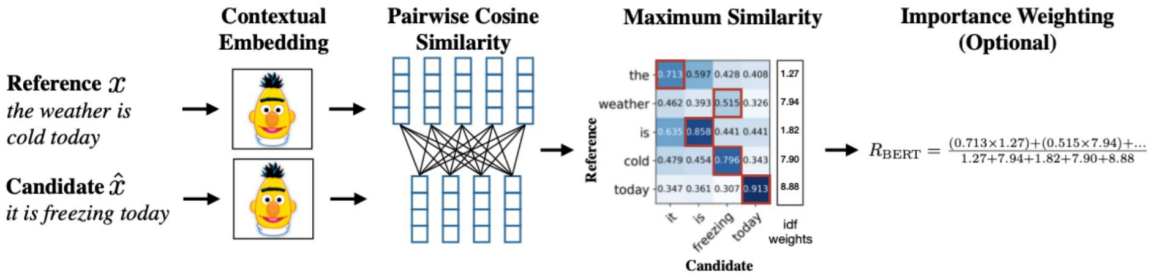


Figure 2.9: The computation of the BERTScore recall metric, R_{BERT}. Source: Zhang et al. [2019b]

Reference:
https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1920-pg-projects/Technical-Writer-Assistant.pdf

Evaluation

- MoverScore
  - Similar to BERTScore, **MoverScore** computes the distance between the embedded candidate and reference summaries. However, **MoverScore** does this by finding the minimum distance one document must be moved in the embedded space to be transformed into the other document

Reference:
https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1920-pg-projects/Technical-Writer-Assistant.pdf

Evaluation

- Meteor
  - The metric is based on the harmonic mean of unigram precision and recall, with recall weighted higher than precision.

Reference:
https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1920-pg-projects/Technical-Writer-Assistant.pdf

Evaluation

Metric	Citations	Year	Contextual Embeddings?	Language Model	n-gram comparison
BERTScore	94	2019	Yes	RoBERTa-lg-MNLI	1-gram
BARTScore	N.A.	N.A.	Yes	BART-lg-MNLI	1-gram
Mover-1	26	2019	Yes	BERT-base-MNLI	1-gram
Mover-2	26	2019	Yes	BERT-base-MNLI	2-gram
BLEURT	3	2020	Yes	BERT-lg	Entire seq.
ROUGE-1	5,124	2004	No	N.A.	1-gram
ROUGE-2	5,124	2004	No	N.A.	2-gram
ROUGE-L	5,124	2004	No	N.A.	Entire seq.

Table 4.1: Side-by-side comparison of each of the evaluation metrics. The fourth and fifth columns indicate if the metric uses contextual embeddings, and the Language Model used to compute these if so. The number of citations is taken from the Google Scholar citations of the original papers introducing the metrics, recorded on 24/08/2020. These papers are: Zhang et al. [2019b], Zhao et al. [2019], Sellam et al. [2020], Lin [2004].

Reference:
https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1920-pg-projects/Technical-Writer-Assistant.pdf

Additional Papers

- BigBird
- DANCER
- Reinforced-selected sentence rewriting

well...I think I will still have some time left...

Recap

Attention Mechanism

- Longformer (sliding window + global attention)
- Efficient attention / GovReport paper (encoder + decoder attention)

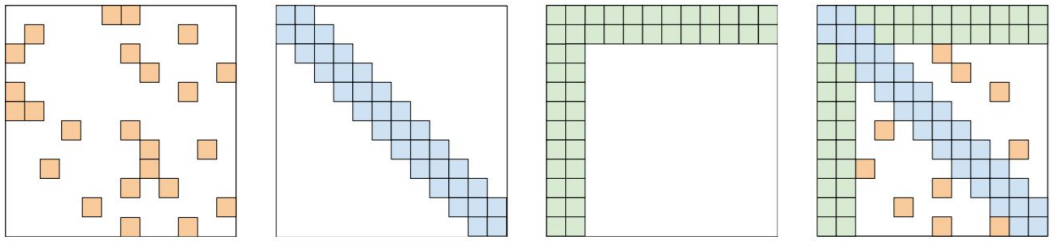
More papers

Attention Mechanism

- Longformer (sliding window + global attention)
- Efficient attention / GovReport paper (encoder + decoder attention)
- BigBird (Sliding Window + Global + random token attention)

https://muppet.fandom.com/wiki/Big_Bird

Credit: “Terence” the **Big** Red **Bird** in Angry Birds Video Game



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Figure 1: Building blocks of the attention mechanism used in BIGBIRD. White color indicates absence of attention. (a) random attention with $r = 2$, (b) sliding window attention with $w = 3$ (c) global attention with $g = 2$. (d) the combined BIGBIRD model.

Credit: BigBird Paper

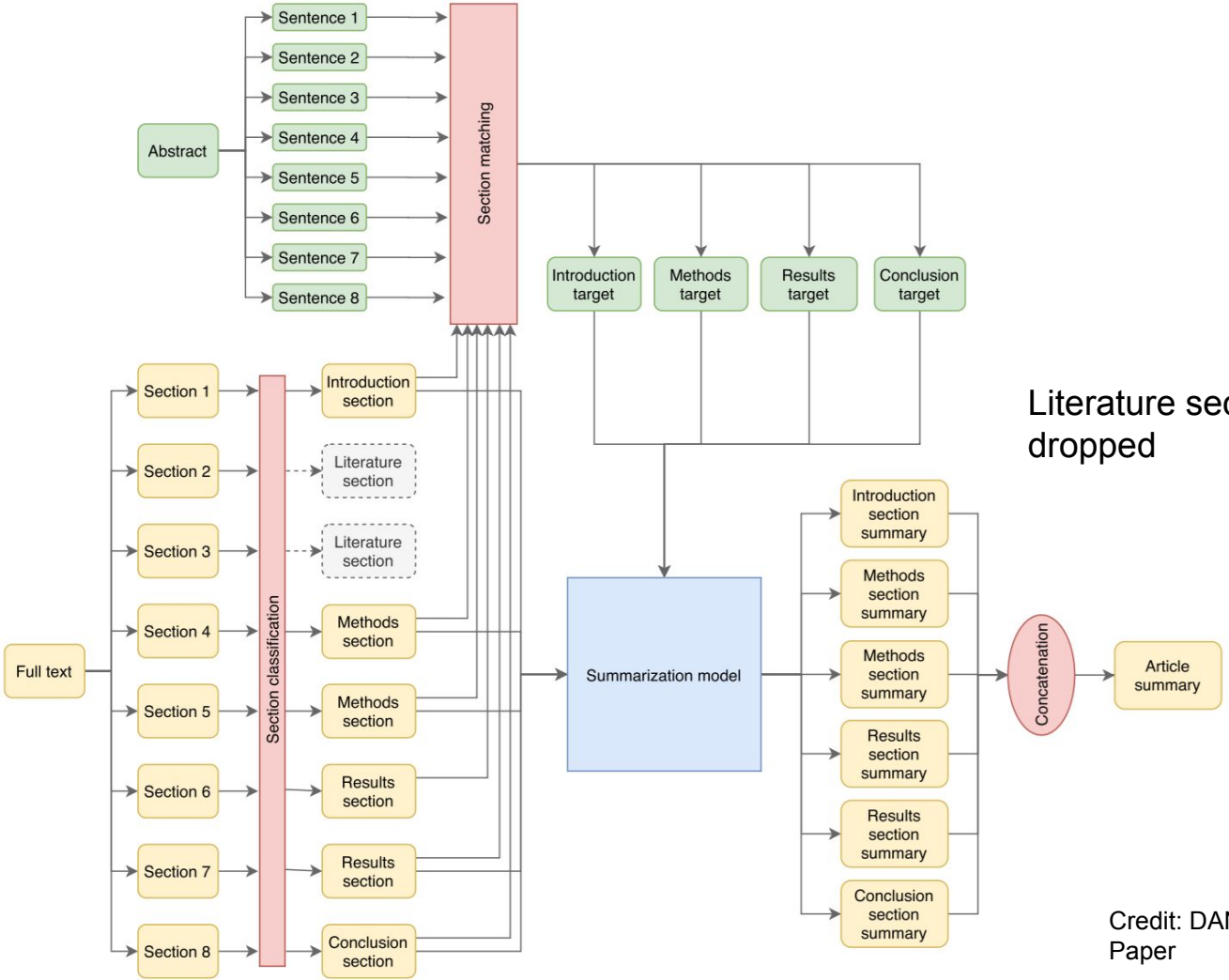
More papers

Divide-and-Conquer Method

- A Divide-and-Conquer Approach to the Summarization of Long Documents, 2020 [Link]
  - **DANCER**
  - Break a document into multiple parts
  - Summarize each part
  - Then combine each part into a full summary

We are following the same approach described in [20] in order to select the sections we want to use for summarization. First we classify each section into different section types like *introduction*, *methods* and *conclusion* based on a heuristic keyword matching of some common keywords in the section header. The specific keywords used for the classification are presented in Table I.

Credit: BigBird Paper



Literature sections are dropped

Credit: DANCER Paper

More papers

Extract-then-abstract Method

- Fast Abstractive Summarization with Reinforce-Selected Sentence Rewriting [link]
- RNN Encoder: context-aware representation
- RNN Decoder: select sentence at a particular timestamp

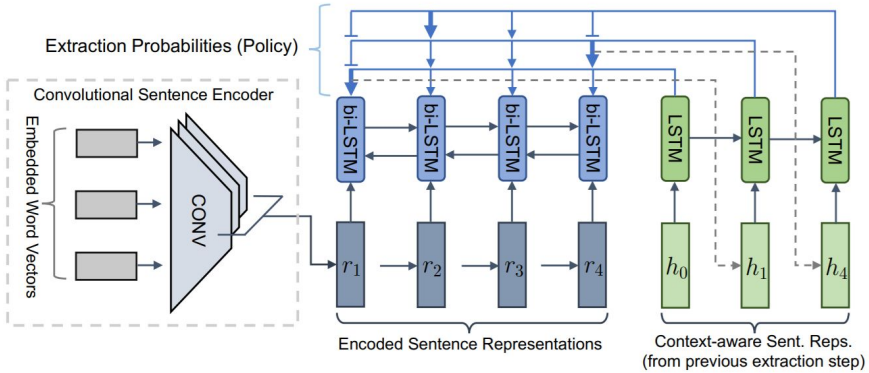


Figure 1: Our extractor agent: the convolutional encoder computes representation r_j for each sentence. The RNN encoder (blue) computes context-aware representation h_j and then the RNN decoder (green) selects sentence j_t at time step t. With j_t selected, h_{j_t} will be fed into the decoder at time $t+1$.

More papers

Extract-then-abstract Method

- **Comparing ground truth summary sentence with generated sentence using Rouge** provides the RL agent with reward

$\pi_{\theta_a,\omega}(c_t, j) = P(j)$ from Eqn. 2 to extract a document sentence and receive a reward[6]

$$r(t+1) = \text{ROUGE-L}_{F_1}(g(d_{j_t}), s_t) \quad (7)$$

after the abstractor summarizes the extracted sentence d_{j_t}. We denote the trainable parameters of the extractor agent by $\theta = \{\theta_a, \omega\}$ for the decoder and hierarchical encoder respectively. We can then train the extractor with policy-based RL. We illus-

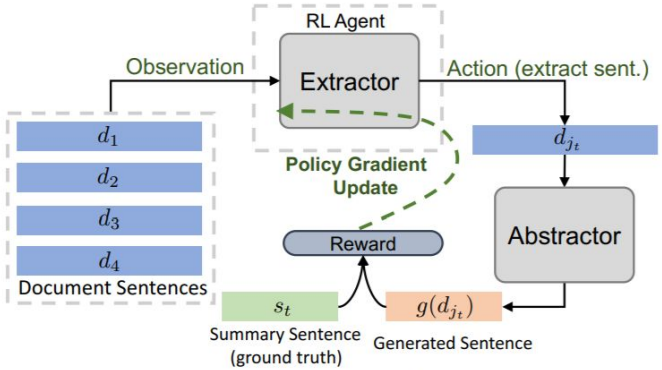


Figure 2: Reinforced training of the extractor (for one extraction step) and its interaction with the abstractor. For simplicity, the critic network is not shown. Note that all d's and s_t are raw sentences, *not* vector representations.

Lastly...

- Conclusions
- Future Directions & Questions
- Backup Slides
    - Reformer
    - Reformer Demo

Conclusion

- Truncating Input to a fixed length
- Sparse Attention (**Longformer, Efficient Attention paper, BigBird**)
- Extract than Abstract Method (**RL-based summarization**)
- Task specific optimizations
    - Such as Divide and Conquer (**Dancer**)
- Hierarchical Models (**HMNet**, **HAT-BART**)

Future Directions & Questions

- How to leverage more on the document structure
  - Meeting: discourse structure
  - Paper: sections
  - Books: chapters, paragraphs, etc.
- What would be a good evaluation metric for long-document summarization
- Query-focused Summarization

More Papers

Modify Attention Mechanism:

- Longformer: The Long-Document Transformer (First Paper), 2020 [link]
- Efficient Attentions for Long Document Summarization (Second Paper), 2021 [link]
- Big Bird: Transformers for Longer Sequences, 2020 [Link]

Extract-then-abstract:

- Long Document Summarization in a Low Resource Setting using Pretrained Language Models, 2021 [Link]

Divide and Conquer:

- A Divide-and-Conquer Approach to the Summarization of Long Documents, 2020 [Link]

Others:

- Extractive Summarization of Long Documents by Combining Global and Local Context, 2019 [Link]
- A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents, 2018 [Link]
- Globalizing BERT-based Transformer Architectures for Long Document Summarization, 2021 [Link]
- HipoRank: Incorporating Hierarchical and Positional Information into Graph-based Unsupervised Long Document Extractive Summarization, 2020 [Link]

Backup Slides

- Reformer

So...

If I could include one more paper

- I recommend the Reformer Paper [link]

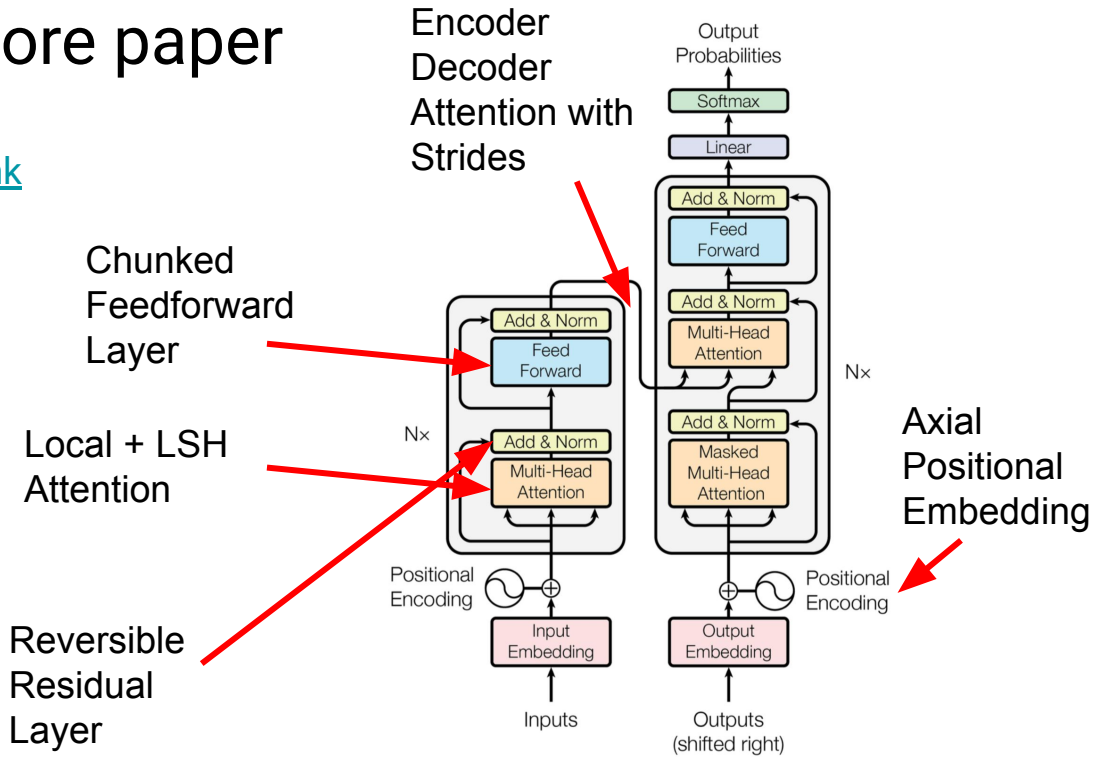


Figure 1: The Transformer - model architecture.

More papers

Reformer

- **Reformer: The Efficient Transformer**

REFORMER: THE EFFICIENT TRANSFORMER

Nikita Kitaev* U.C. Berkeley & Google Research kitaev@cs.berkeley.edu

Łukasz Kaiser* Google Research

Anselm Levskaya Google Research

{lukaszkaiser,levskaya}@google.com

ABSTRACT

Large Transformer models routinely achieve state-of-the-art results on a number of tasks but training these models can be prohibitively costly, especially on long sequences. We introduce two techniques to improve the efficiency of Transformers. For one, we replace dot-product attention by one that uses locality-sensitive hashing, changing its complexity from O(L^2) to O($L \log L$), where L is the length of the sequence. Furthermore, we use reversible residual layers instead of the standard residuals, which allows storing activations only once in the training process instead of N times, where N is the number of layers. The resulting model, the Reformer, performs on par with Transformer models while being much more memory-efficient and much faster on long sequences.

More papers

- The memory improvements can be attributed to 4 features which the Reformer authors introduced to the transformer world:
  - **Reformer Self-Attention Layer**
  - **Chunked Feed Forward Layers**
  - **Reversible Residual Layers**
  - **Axial Positional Encodings**

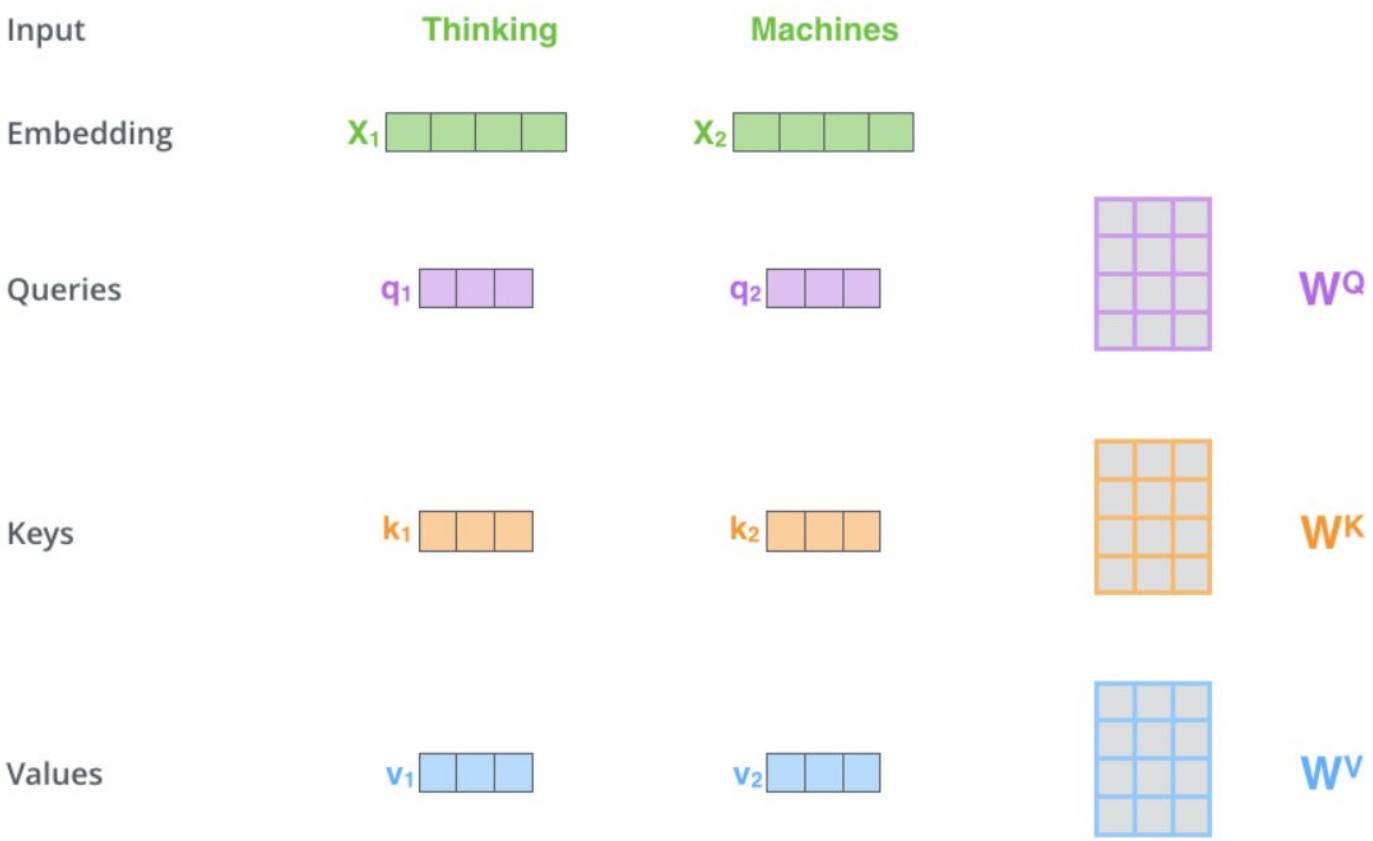
More papers

As discussed by Zhangir:

for each word, we create a Query vector, a Key vector, and a Value vector. These vectors are created by multiplying the embedding by three matrices that we trained during the training process.

self-attention layer projects takes query, key and value matrices Q,K,V and computes the output Z using the softmax operation as follows:

$$Z=\text{softmax}(QK^T) V$$



Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

More papers

- The memory improvements can be attributed to 4 features which the Reformer authors introduced to the transformer world:
  - **Reformer Self-Attention Layer**
    - LSH Attention

More papers

The idea behind **LSH self-attention** is based on the insight that if n is large, the softmax applied on the QK^T attention dot-product **weighs only very few value vectors with values significantly larger than 0 for each query vector**.

This owes to the fact that the softmax function puts exponentially more weight on larger input values.

The next problem is to find the vectors k that have high cosine similarity with q_i for all i.

For models with LSH attention, we want queries and keys (Q and K) to be identical.

Result ?

More papers

Sharing the query and key projections: Set Q=K does not impact the performance of a transformer

Therefore, the query vectors can be clustered into buckets, such that all query vectors that belong to the same bucket have a high cosine similarity to each other

For each set of indices, the softmax function on the corresponding bucket of query vectors approximates the softmax function of global self-attention with shared query and key projections for all position indices

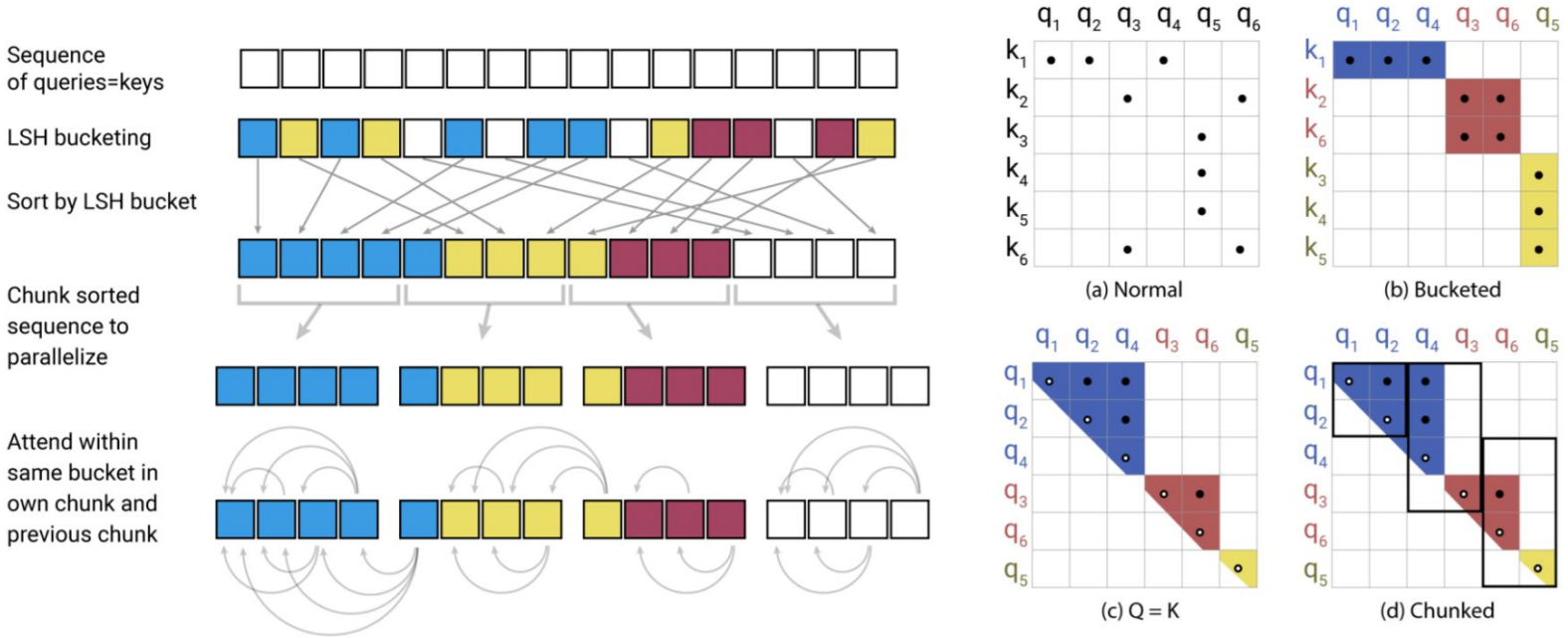
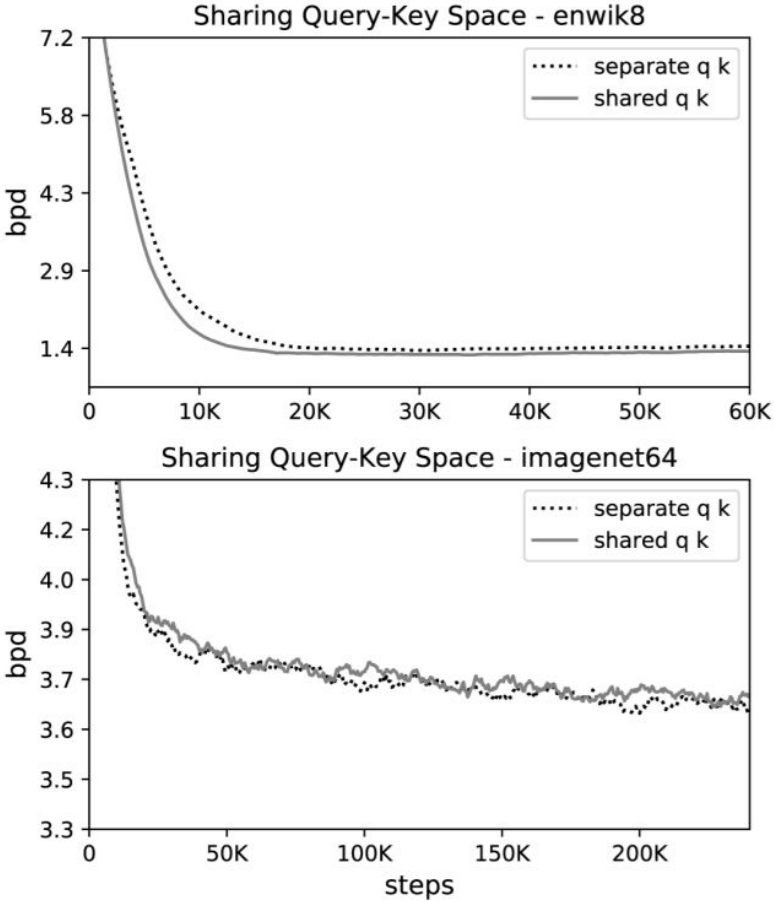


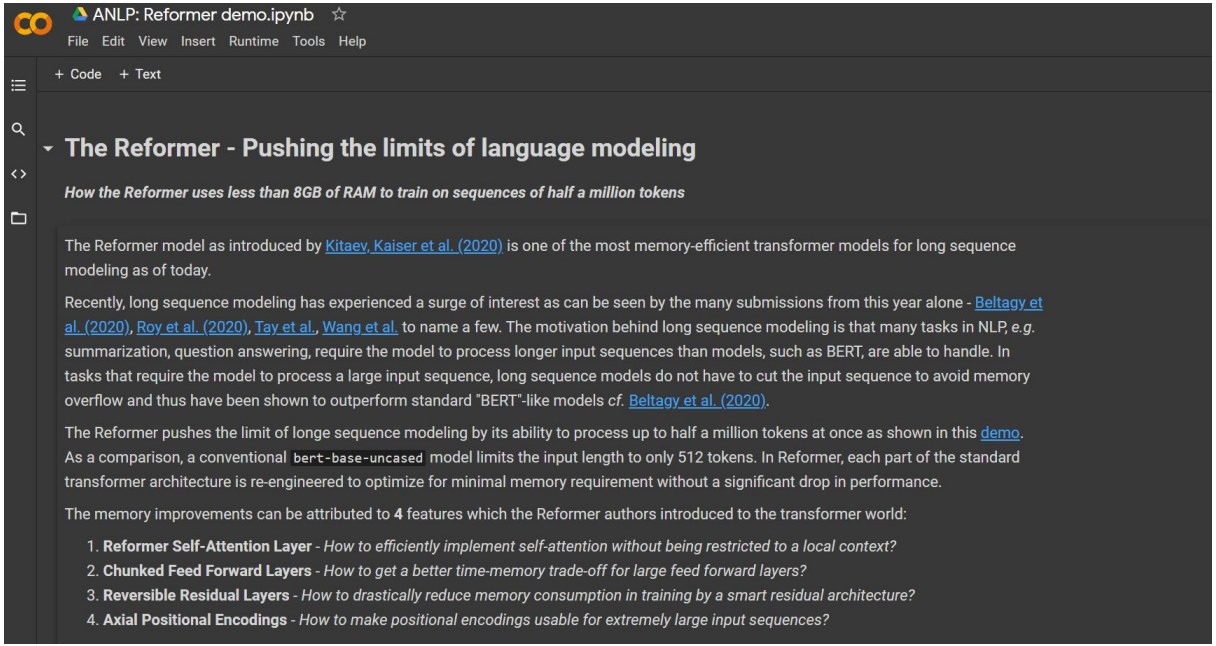
Figure 2: Simplified depiction of LSH Attention showing the hash-bucketing, sorting, and chunking steps and the resulting causal attentions. (a-d) Attention matrices for these varieties of attention.



Effect of sharing QK. We first consider the effect of shared-QK attention on a regular Transformer model. Shared-QK attention sets $k_j = \frac{q_j}{\|q_j\|}$ and prevents tokens from attending to themselves (except when no other context is available). In the left part of Figure 3, we plot perplexity curves for both regular and shared-QK attention. A shared query-key space does not perform worse than regular attention; in fact, for enwik8 it appears to train slightly faster. In other words, we are not sacrificing accuracy by switching to shared-QK attention.

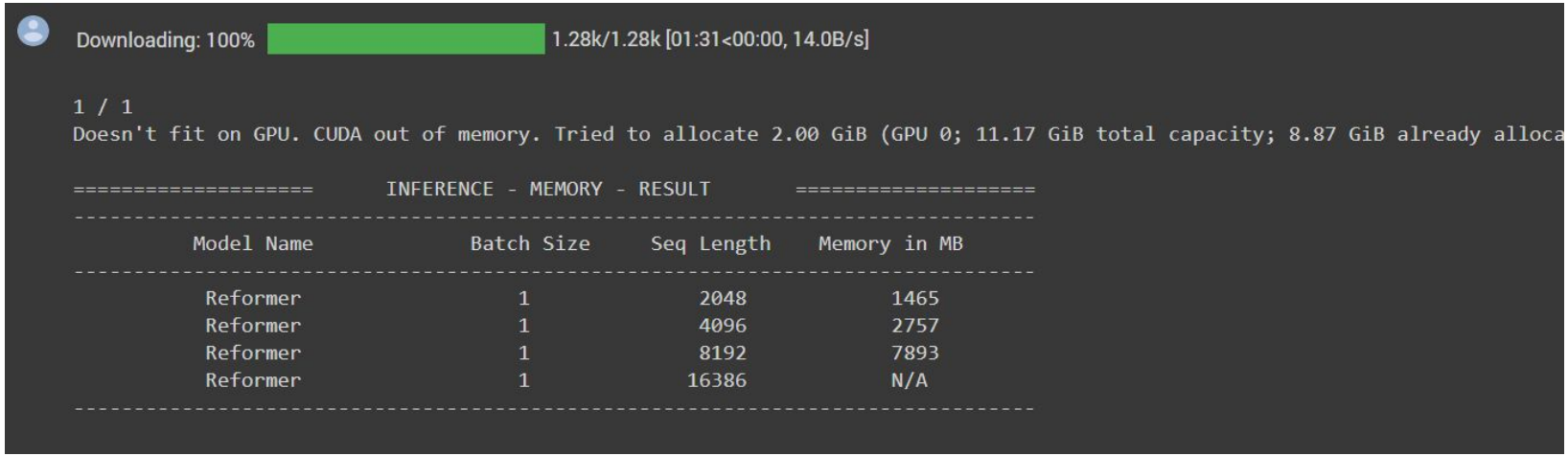
Google Colab Notebook for Demonstration

https://colab.research.google.com/drive/1-F6OvjozIBP4xxY0Se47XJpRyw_qqE2e#scrollTo=mk1ETLlfEMGA



Google Colab Notebook for Demonstration

Full Attention

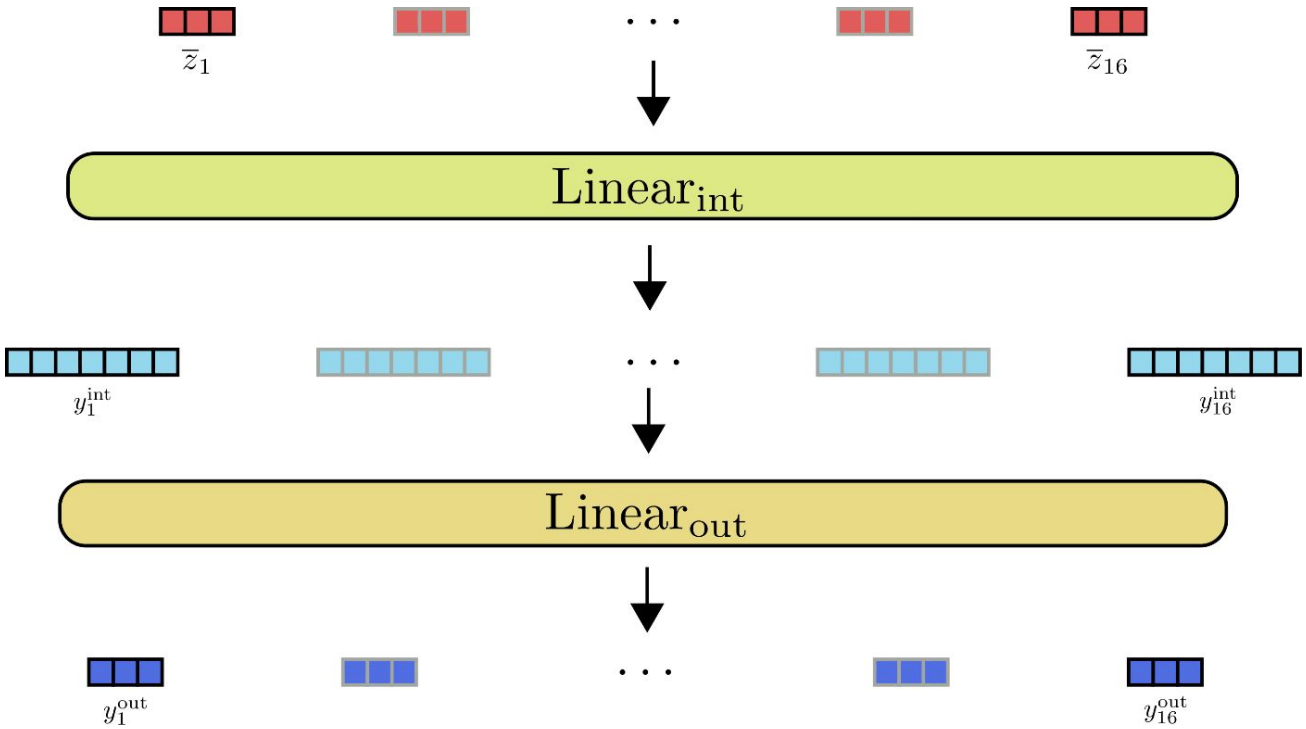


Google Colab Notebook for Demonstration

Local + LSH Attention

```
====================      INFERENCE - MEMORY - RESULT       ====================
--------------------------------------------------------------------------------
          Model Name             Batch Size     Seq Length    Memory in MB
--------------------------------------------------------------------------------
           Reformer                    1            2048           1785
           Reformer                    1            4096           2621
           Reformer                    1            8192           4281
           Reformer                    1           16384           7607
           Reformer                    1           32768           N/A
           Reformer                    1           65436           N/A
--------------------------------------------------------------------------------
```

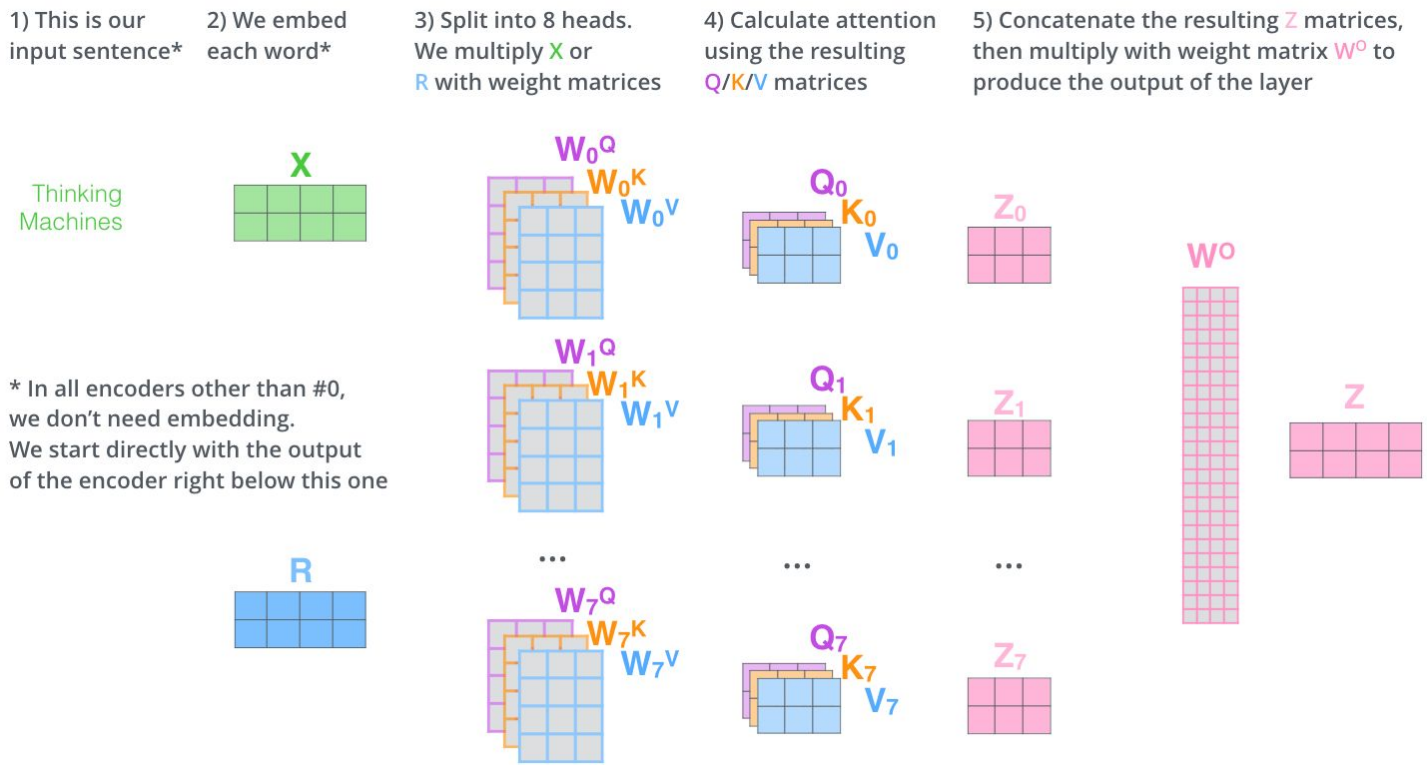
Chunked Feed Forward Layer in Transformer

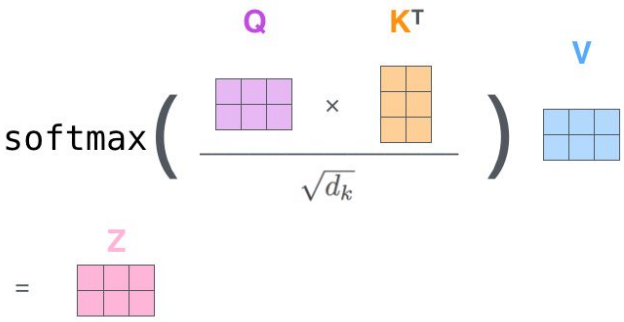


Chunked Feed Forward Layer in Transformer

The author observes that the intermediate vector dimension is usually larger than the output vector dimension!

Chunked Feed Forward Layer in Transformer



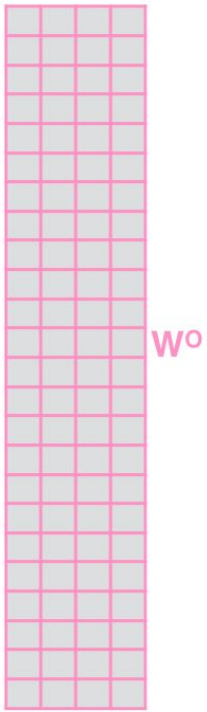

Q
K^T
V
softmax
$\sqrt{d_k}$
Z
=

Chunked Feed Forward Layer in Transformer

1) Concatenate all the attention heads

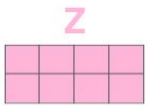
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



To trade memory for speed, one can thus chunk the linear layers computation to only process one chunk at the time.

Google Colab Notebook for Demonstration

Chunk vs Non-chunk

Difference can only be seen with sequence token count > 4096

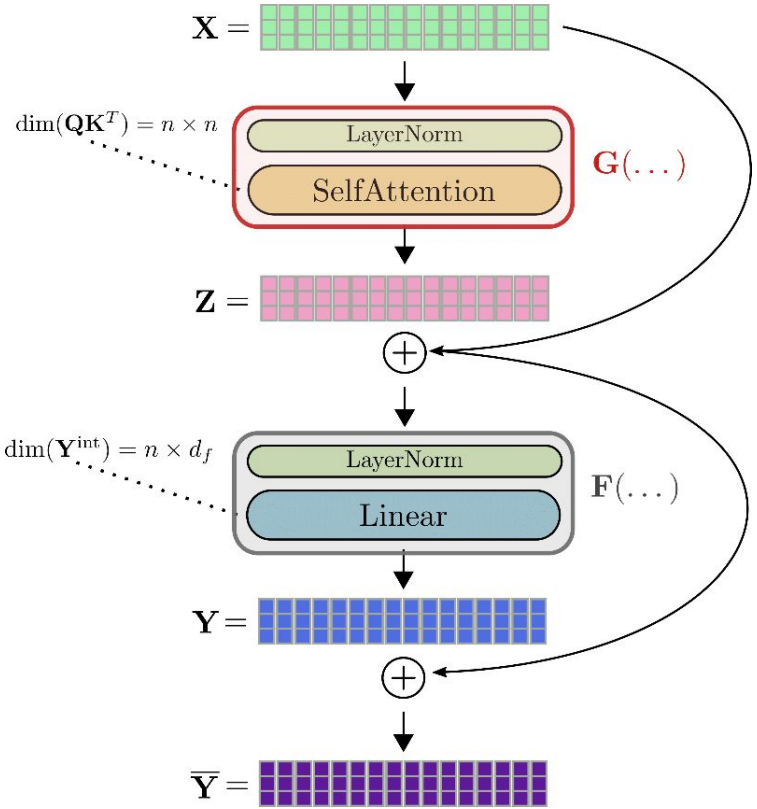
```
====================      INFERENCE - MEMORY - RESULT      ====================
--------------------------------------------------------------------------------
          Model Name             Batch Size     Seq Length    Memory in MB
--------------------------------------------------------------------------------
      Reformer-No-Chunk                8             1024            3743
      Reformer-No-Chunk                8             2048            5539
      Reformer-No-Chunk                8             4096            9087
        Reformer-Chunk                 8             1024            2973
        Reformer-Chunk                 8             2048            3999
        Reformer-Chunk                 8             4096            6011
--------------------------------------------------------------------------------
```

Reversible Residual Layers

Recap: why training a model requires much more memory than the inference of the model.

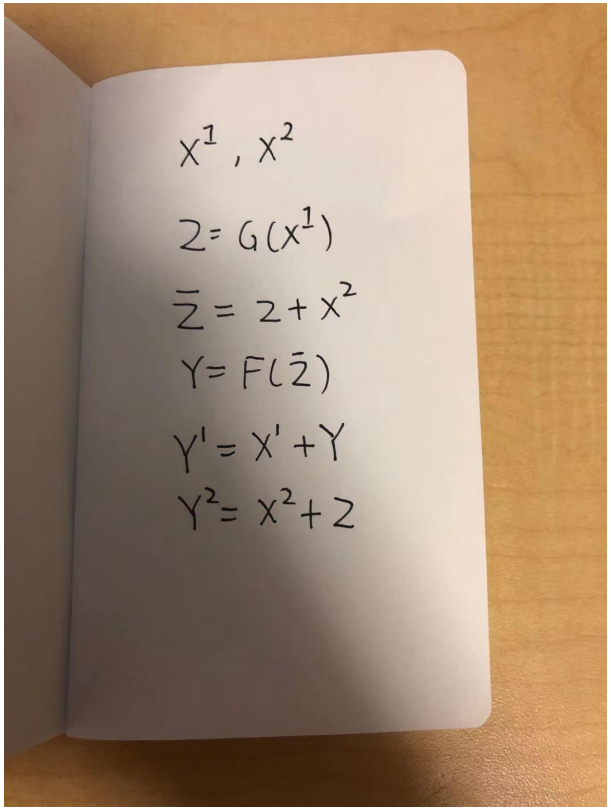
When running a model in inference, the required memory equals more or less the memory it takes to compute the single largest tensor in the model. On the other hand, when training a model, the required memory equals more or less the sum of all differentiable tensors.

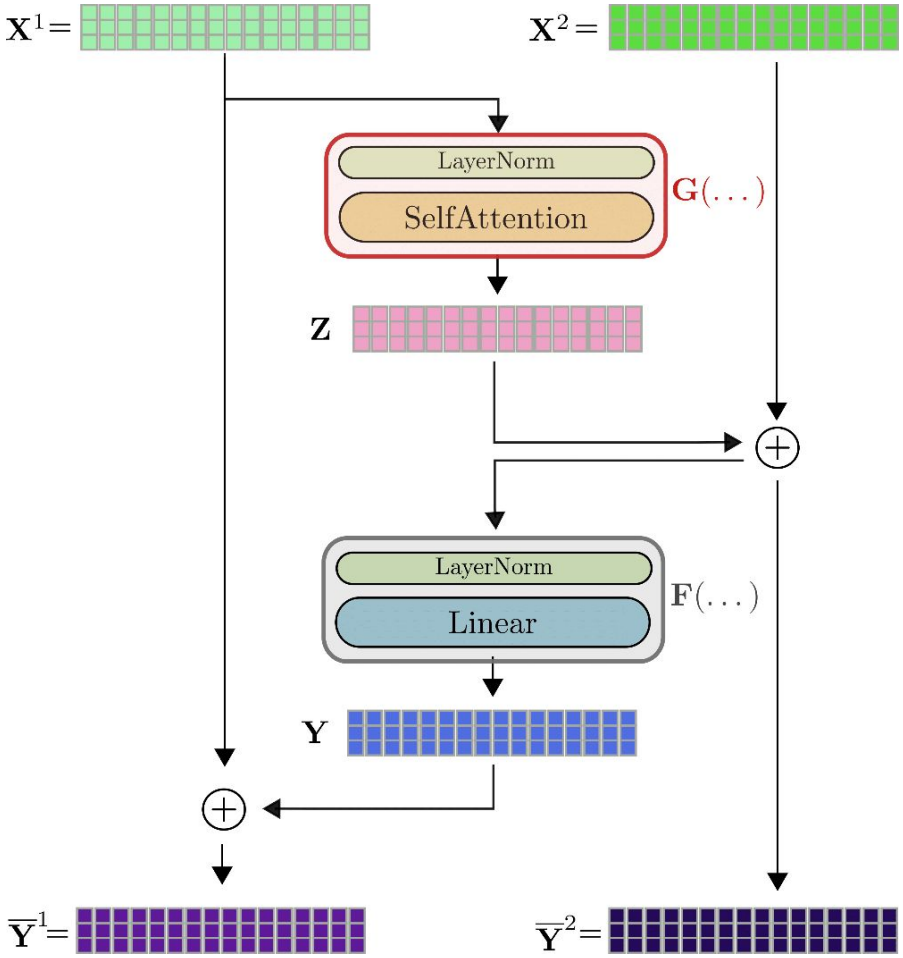
Reversible Residual Layers



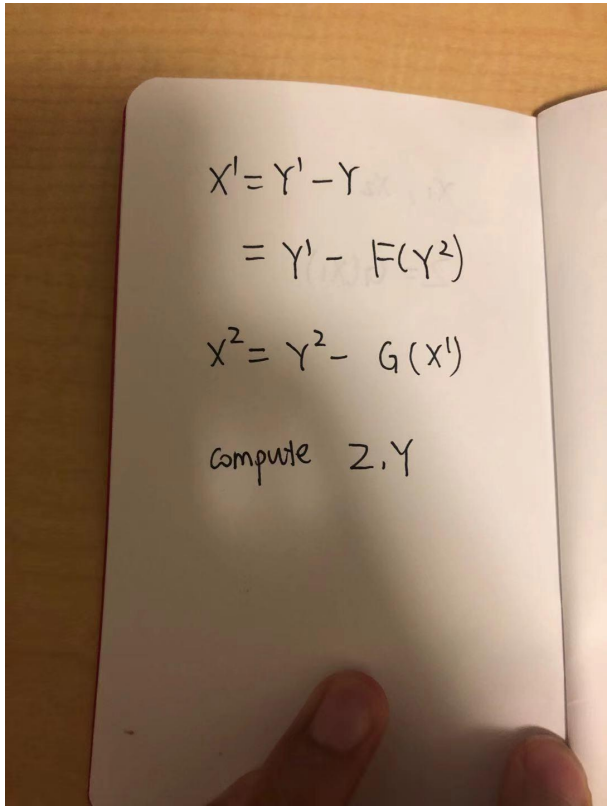
While ∂Z can be calculated on-the-fly and discarded afterward, the values for Z and X have to be calculated and stored during the forward pass since it is not possible to recalculate them easily on-the-fly during backpropagation.

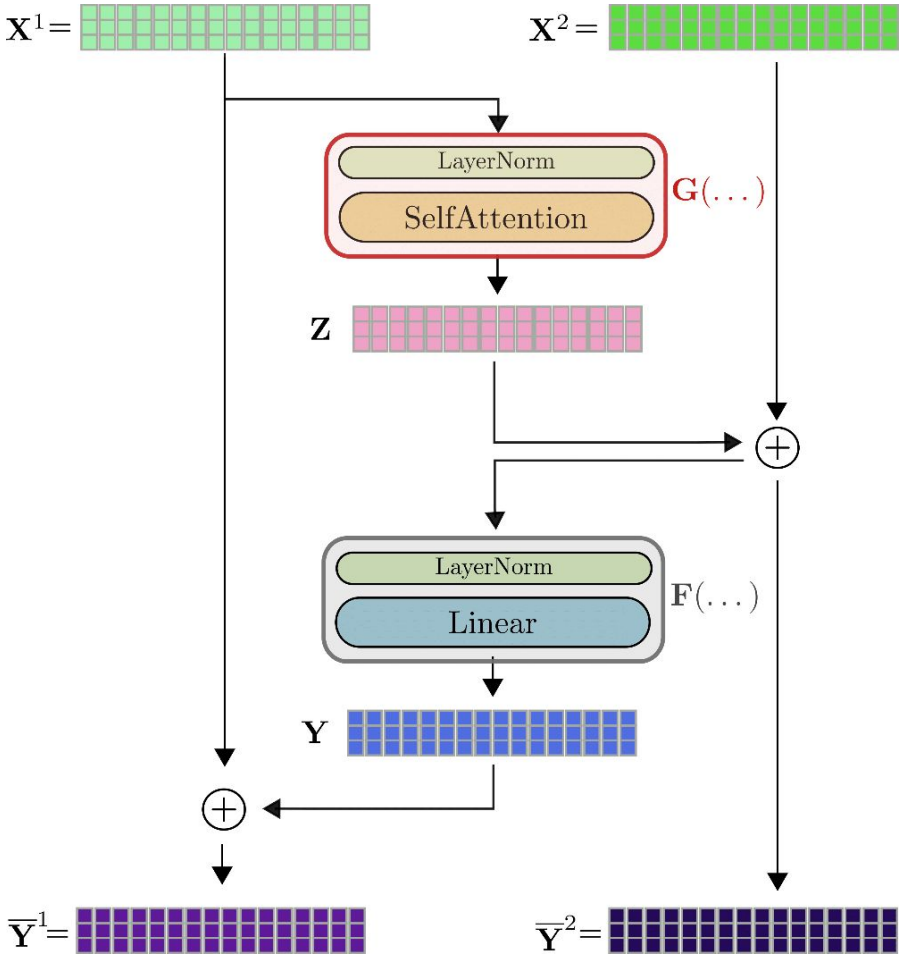
Reversible Residual Layers





Reversible Residual Layers





Reversible Residual Layers

In summary:

The overhead of 2 forward pass of G and F is traded against not having to store any activations during the forward pass!

Google Colab

```
====================        TRAIN - MEMORY - RESULTS        ====================
--------------------------------------------------------------------------------
          Model Name             Batch Size     Seq Length    Memory in MB
--------------------------------------------------------------------------------
        Bert-4-Layers                8              512            4103
        Bert-8-Layers                8              512            5759
        Bert-12-Layers               8              512            7415
--------------------------------------------------------------------------------
```

```
====================        TRAIN - MEMORY - RESULTS        ====================
--------------------------------------------------------------------------------
          Model Name             Batch Size     Seq Length    Memory in MB
--------------------------------------------------------------------------------
      Reformer-4-Layers              8              512            4607
      Reformer-8-Layers              8              512            4987
      Reformer-12-Layers             8              512            5367
--------------------------------------------------------------------------------
```

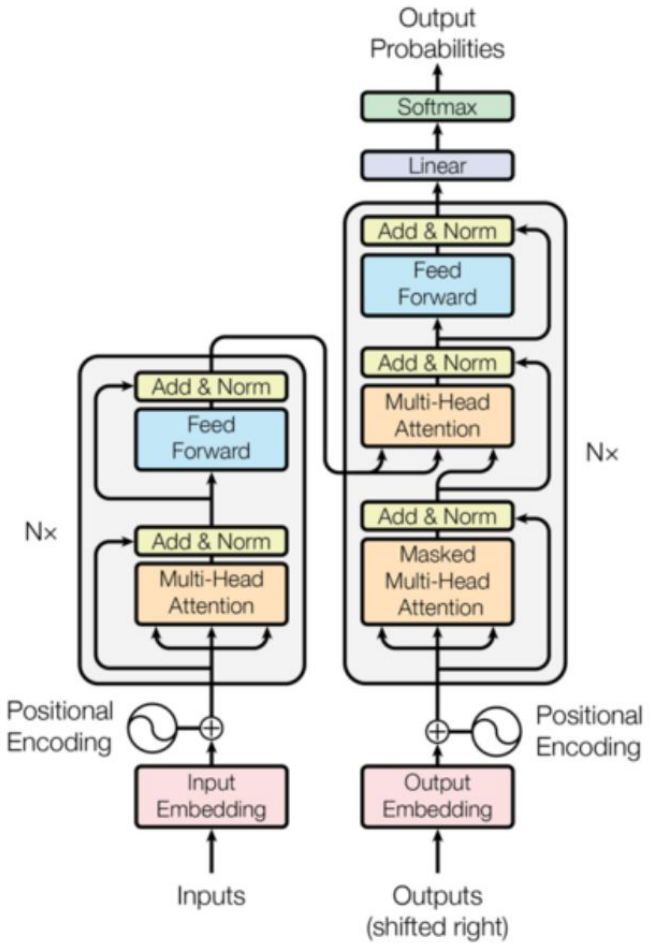
What is positional embedding

As each word in a sentence simultaneously flows through the Transformer’s encoder/decoder stack, The model itself doesn’t have any sense of position/order for each word

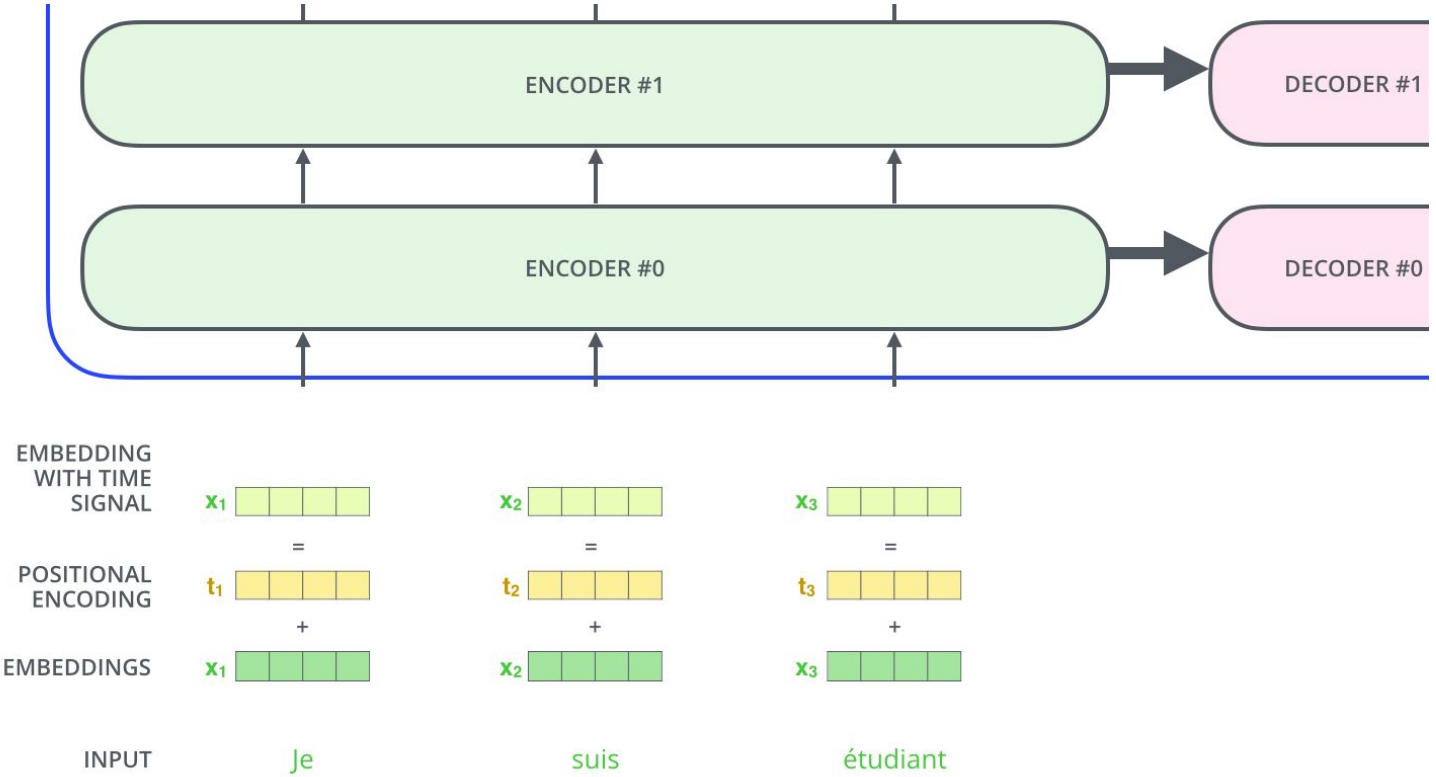
One possible solution to give the model some sense of order is to add a piece of information to each word about its position in the sentence. We call this “piece of information”, the positional encoding.

Axial Positional Embedding

Ith positional embedding is simply added to the ith input vector

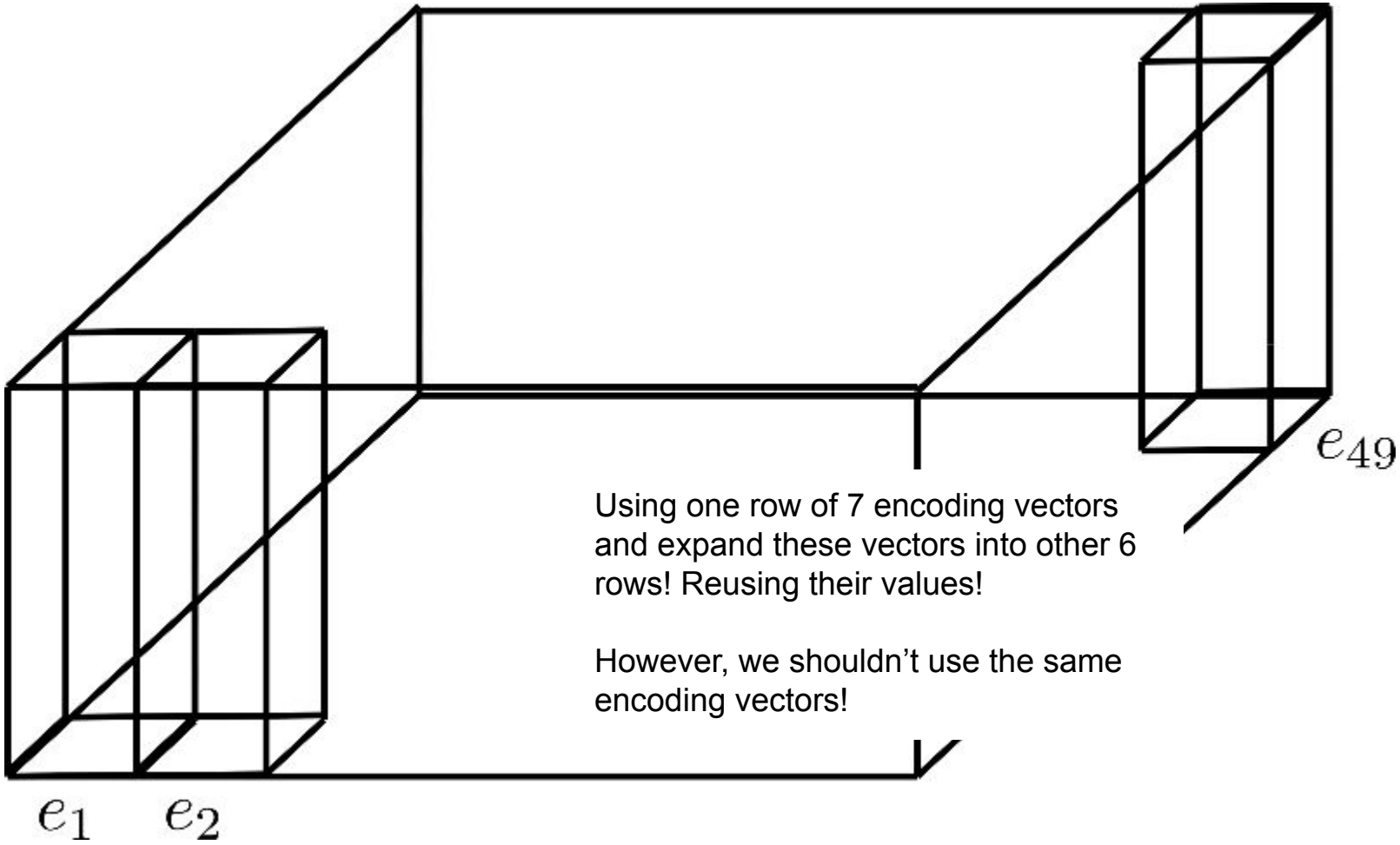


Axial Positional Embedding



Axial Positional Embedding

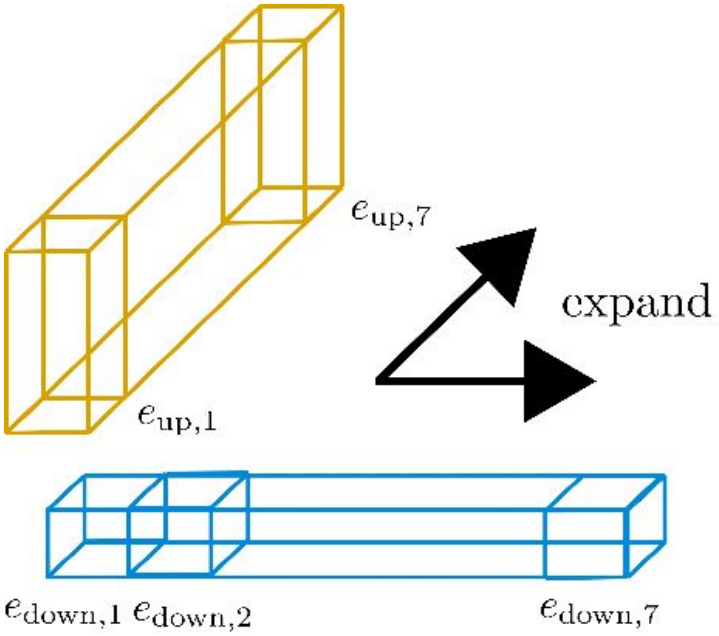
Let's imagine, we want to train a Reformer model on sequences of a length of up to 0.5M tokens and an input vector config.hidden_size of 1024. The corresponding positional embeddings have a size of 0.5M×1024~512M parameters, which corresponds to a size of 2GB.

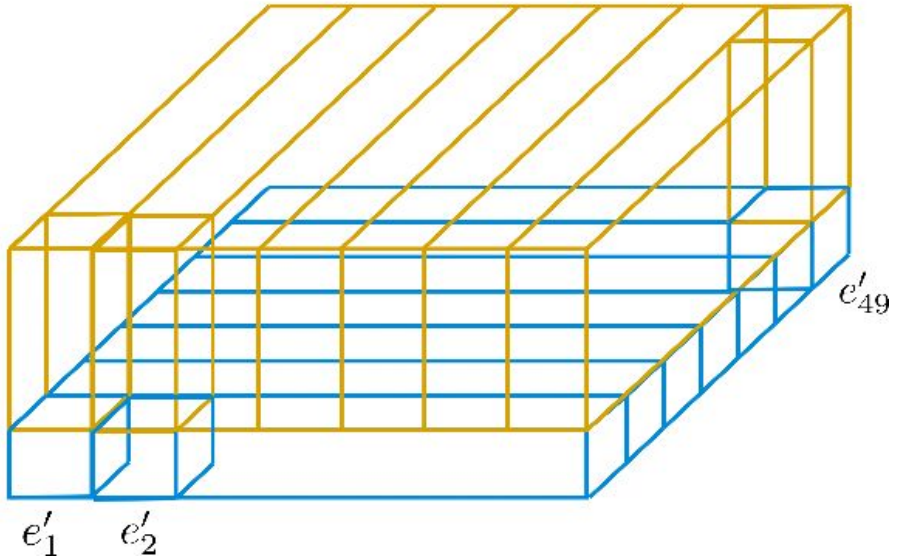


Using one row of 7 encoding vectors and expand these vectors into other 6 rows! Reusing their values!

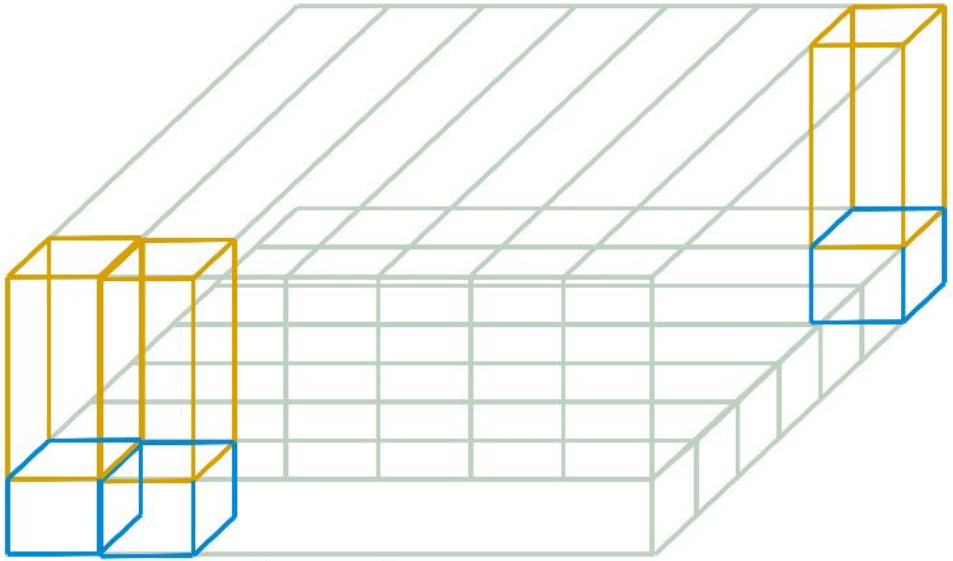
However, we shouldn't use the same encoding vectors!

Axial Positional Embedding





Axial Positional Embedding



$$e'_{49} = \begin{bmatrix} e_{\text{up},7} \\ e_{\text{down},7} \end{bmatrix}$$

$$e'_1 = \begin{bmatrix} e_{\text{up},1} \\ e_{\text{down},1} \end{bmatrix} \quad e'_2 = \begin{bmatrix} e_{\text{up},1} \\ e_{\text{down},2} \end{bmatrix}$$

Now it should be more understandable how the final positional encoding vectors $\mathbf{E}'$ are calculated only from $\mathbf{E}_{\text{down}}$ of dimension $d_h^1 \times n_{\text{max}}^1$ and $\mathbf{E}_{\text{up}}$ of dimension $d_h^2 \times n_{\text{max}}^2$.

$$\mathbf{e'}_i = \begin{bmatrix} \mathbf{e}_{\text{down},\, i \% n^1_{\text{max}}} \\ \mathbf{e}_{\text{up},\, \left\lfloor \frac{i}{n^2_{\text{max}}} \right\rfloor} \end{bmatrix}$$

whereas $n^1_{\text{max}} = 7$ and $n^2_{\text{max}} = 7$ in our example. These new encodings $\mathbf{E'} = [\mathbf{e'}_1, \ldots, \mathbf{e'}_{n_{\text{max}}}]$ are called **Axial Position Encodings**.

None of the axial positional embedding is the same by design!

Axial Positional Embedding

Overall Reduction:

From 2 GB positional encoding to 3 MB

$$n_{\max} \times d_h \text{ to } n_{\max}^1 \times d_h^1 + n_{\max}^2 \times d_h^2.$$

That’s it!

Thank you!

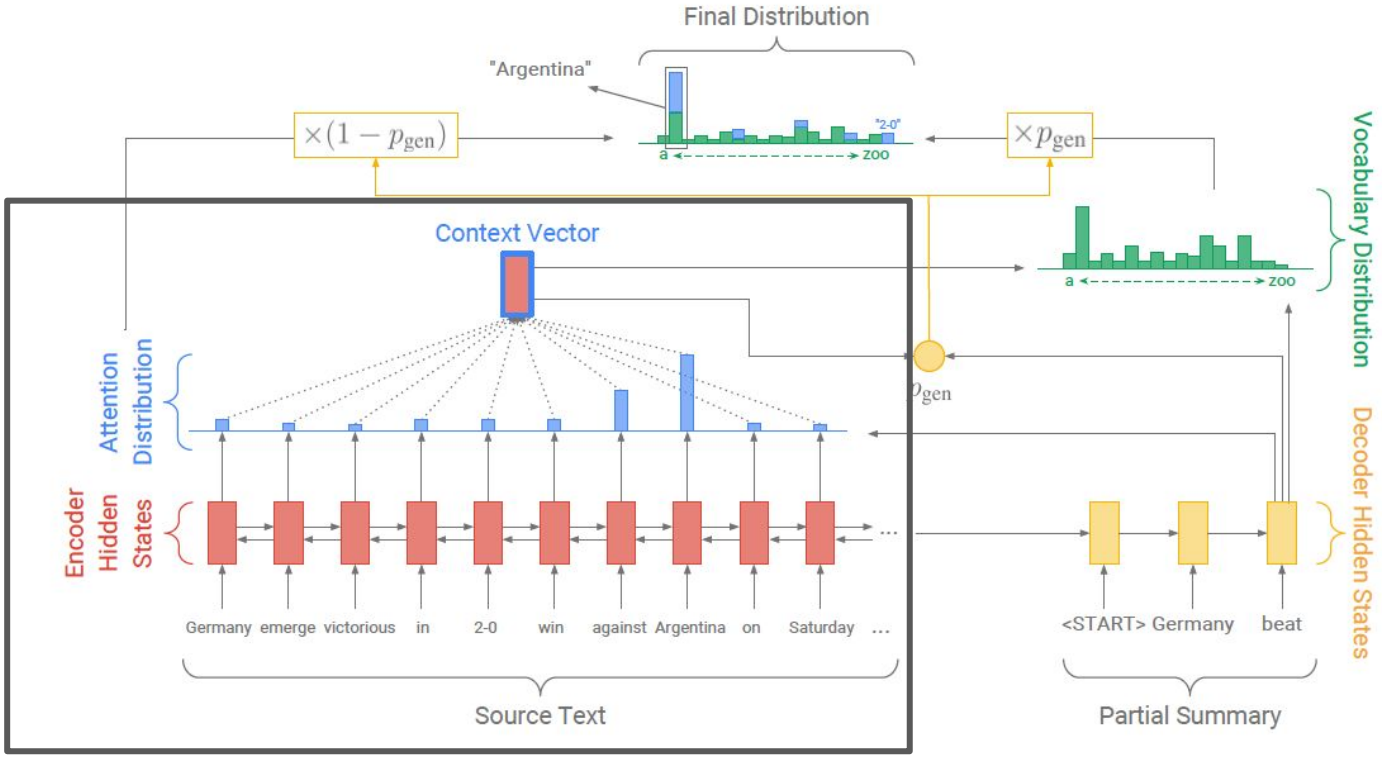
23/9/2021

Additional Papers

Long Document Summarization

- Longformer: The Long-Document Transformer (First Paper), 2020 [link]
- Efficient Attentions for Long Document Summarization (Second Paper), 2021 [link]
- Additional papers:
  - Big Bird: Transformers for Longer Sequences, 2020 [Link]
  - A Divide-and-Conquer Approach to the Summarization of Long Documents, 2020 [Link]
  - Extractive Summarization of Long Documents by Combining Global and Local Context, 2019 [Link]
  - A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents, 2018 [Link]

Introduction



[link]