Effective transformers

Skoltech

Alexey Zaytsev

Some slides are from Zabolotnyi Artem

Transformers recap and their limitations

Skoltech

Why transformer architecture?

Replace RNN in and become SOTA in NLP tasks

- Text classification
- Machine translation
- Text summarization

Key advantages:

- Do not have a recurrent dependency
- Train required a significant amount of data, but you can use **pre-trained models** and fine-tune for certain a task

Skoltech

Key value interpretation

$\boldsymbol{q}_i$ - query to a database	**Hidden state of the decoder**
$\boldsymbol{k}_j$ - keys in the database	Hidden state of the encoder
$\boldsymbol{v}_j$ - values in the database	Hidden state of the encoder

1. Calculate the attention scores

$\alpha_j = e(\boldsymbol{q}_i, \boldsymbol{k}_j) = \boldsymbol{s}_i^T \boldsymbol{k}_j$

$\boldsymbol{\alpha} = \mathrm{softmax}(\boldsymbol{\alpha})$

2. Extract the information as the weighted sum of values

$\mathbf{a}_\mathrm{i} = \sum_{j=1}^{T_x} \alpha_j \boldsymbol{v}_j$

Skoltech

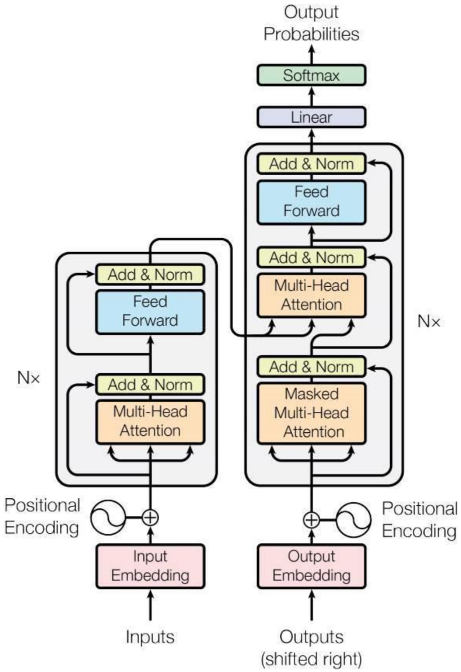
Matrix key value interpretation

$\boldsymbol{q}_i$ - query to a database	**Hidden state of the decoder**
$\boldsymbol{k}_j$ - keys in the database	Hidden state of the encoder
$\boldsymbol{v}_j$ - values in the database	Hidden state of the encoder

We calculate correspondences

$$A(q, K, V) = \sum_i \frac{\exp(\boldsymbol{q}_i^T \boldsymbol{k}_j)}{\sum_l \exp(\boldsymbol{q}_i^T \boldsymbol{k}_l)} \boldsymbol{v}_j$$

$$A(Q, K, V) = \mathrm{softmax}(\mathrm{QK^T})\mathrm{V}$$

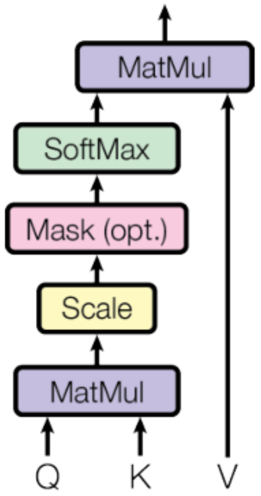


Skoltech

Attention / Self-attention block

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

Scaled Dot-Product Attention



query key value

d_k is the dimension of query and key,
we scale to take control of large values of dot-product in high dimensions

A possible option is to replace scaled dot-product used here with additive attention: a single-hidden layer neural network.

“Attention is all you need” paper

Self-attention block

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

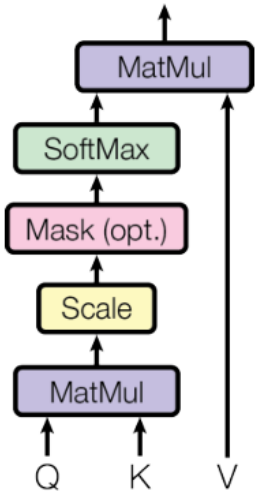
We produce queries, keys, and values
using initial word embeddings for a sequence of length $\mathrm{d_x}$

$$\mathrm{Q} = \mathrm{X\,W^Q}, \dim(\mathrm{W^Q}) = \mathrm{d_x} \times \mathrm{d}_q,$$

$$\mathrm{K} = \mathrm{X\,W}^K, \dim(\mathrm{W^Q}) = \mathrm{d_x} \times \mathrm{d}_k,$$

$$\mathrm{V} = \mathrm{X\,W}^V, \dim(\mathrm{W^Q}) = \mathrm{d_x} \times \mathrm{d}_v,$$

Scaled Dot-Product Attention



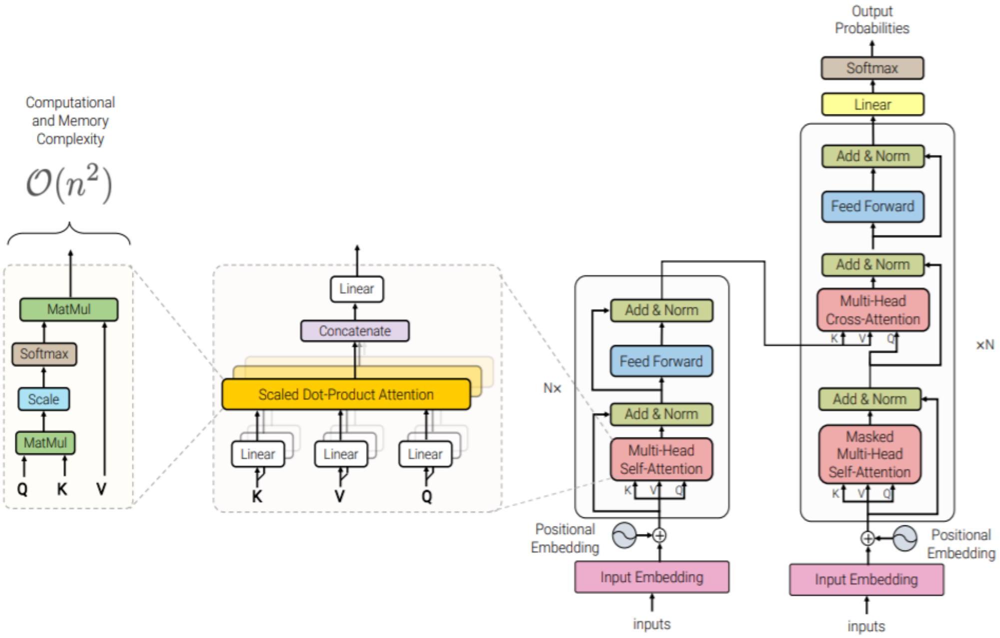
query key value

“Attention is all you need” paper

Self-attention block complexity

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

Memory complexity is $O(d_x^2)$
Computational complexity is $O(d_x^2)$



Skoltech

Self-attention block complexity

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

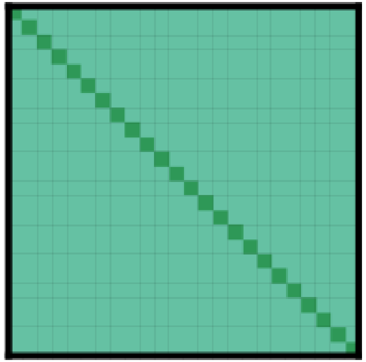
Memory complexity is $O(d_x^2)$
Computational complexity is $O(d_x^2)$

Max sequence size in popular models (e.g. BERT) is only $d_x = 512$ tokens

In modern models tokens are parts of words

Output

Input



Full n^2 attention

Skoltech

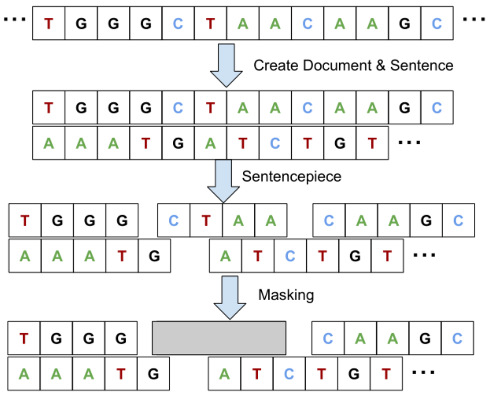
Sequence length in real problems

Summarization: 781 tokens in CNN / Daily Mail dataset [1]

Promoter Region Prediction from DNA: about 4k tokens in the dataset [2]

Bank Transactions: more than 1k tokens during one year [3]

<u>All lengths are larger than 512!</u>



Skoltech

1. Hermann, Karl Moritz, et al. "Teaching machines to read and comprehend." *Advances in neural information processing systems* 28 (2015).
2. Zaheer, M. et al. (2020). Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, *33*.
3. Personal experience

A simple possible solution

1. Split to blocks

2. Calculate embedding for each block

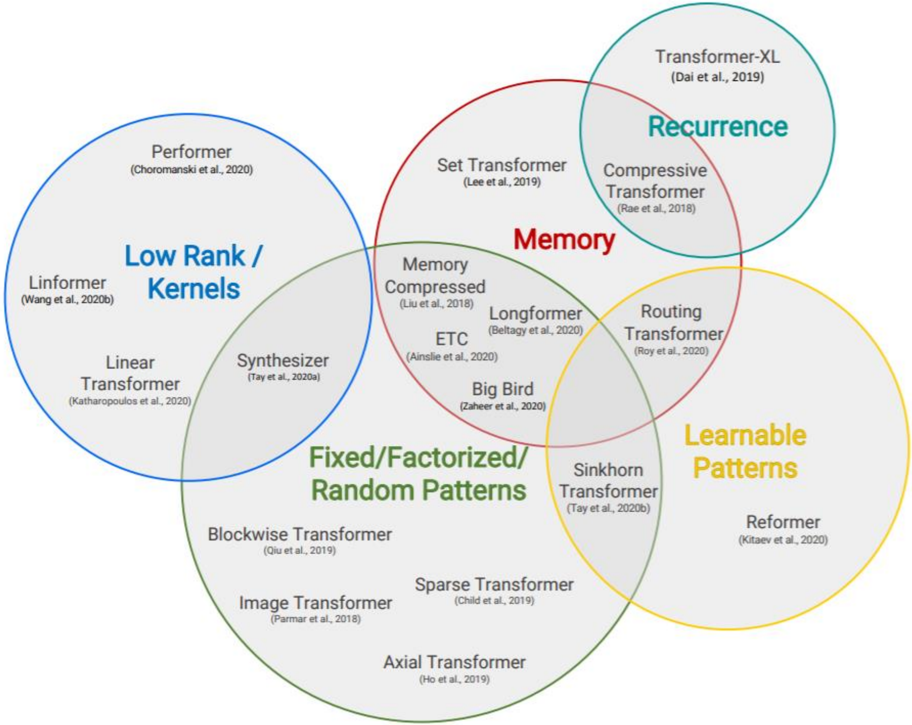
3. Unite these embeddings (an heuristic)

- Not natural
- Additional hyperparameters
- We loss cross-sentence context

Skoltech

Transformer for long sequences

To work with sequences with significant length we should decrease memory consumption and computation complexity $O(n^2)$



Skoltech

Tay, Yi, et al. "Efficient transformers: A survey." *arXiv preprint arXiv:2009.06732* (2020).

Other solutions

Attention matrix:
LTR – proceed in chunks
Sparse – use sparse attention pattern

char-LM:
character level LM

pretrain:
start from other models

Model	attention matrix	char-LM	other tasks	pretrain
Transformer-XL (2019)	ltr	yes	no	no
Adaptive Span (2019)	ltr	yes	no	no
Compressive (2020)	ltr	yes	no	no
Reformer (2020)	sparse	yes	no	no
Sparse (2019)	sparse	yes	no	no
Routing (2020)	sparse	yes	no	no
BP-Transformer (2019)	sparse	yes	MT	no
Blockwise (2019)	sparse	no	QA	yes
Our Longformer	sparse	yes	multiple	yes

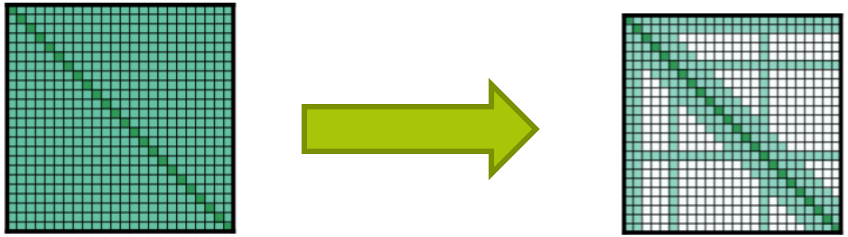
Skoltech

Fixed patterns

The earliest modifications to self-attention simply sparsifies the attention matrix by limiting the field of view to fixed, predefined patterns such as local windows and block patterns of fixed strides.

Conclusion:

- Not too hard to implement
- Fast
- Chosen pattern of attention matrix could be not optimal for data

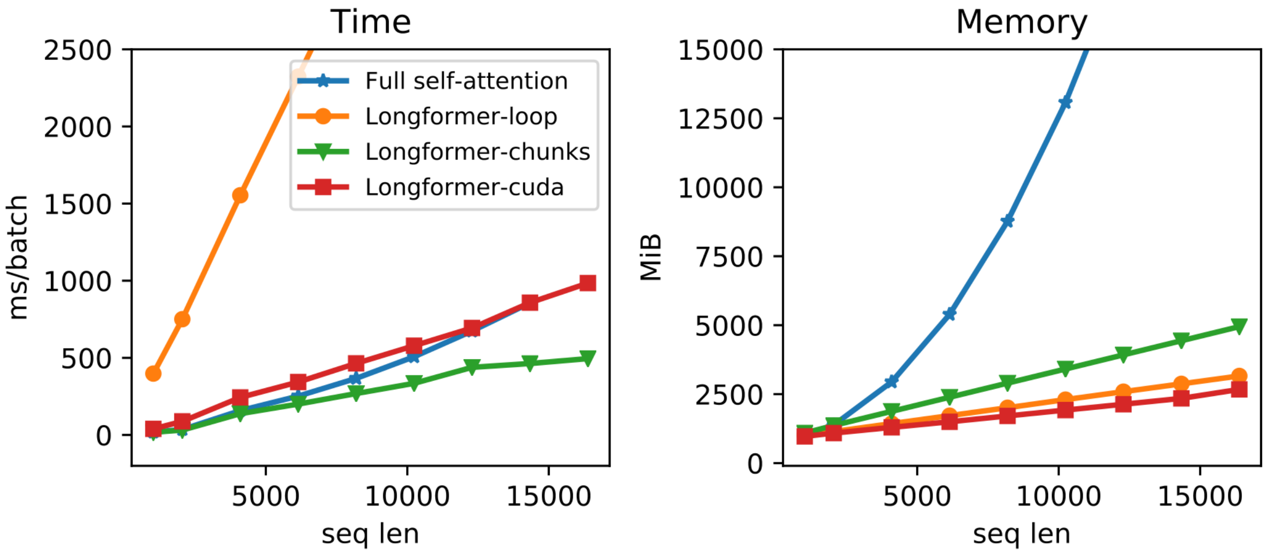


(a) Full n^2 attention

Longformers

Longformers performance

For Full self-attention we have <u>quadratic</u> memory scaling
For Longformer-chunks we have <u>linear</u> memory scaling



Skoltech

Longformer attentions

Memory requirements,
n is the sequence length:

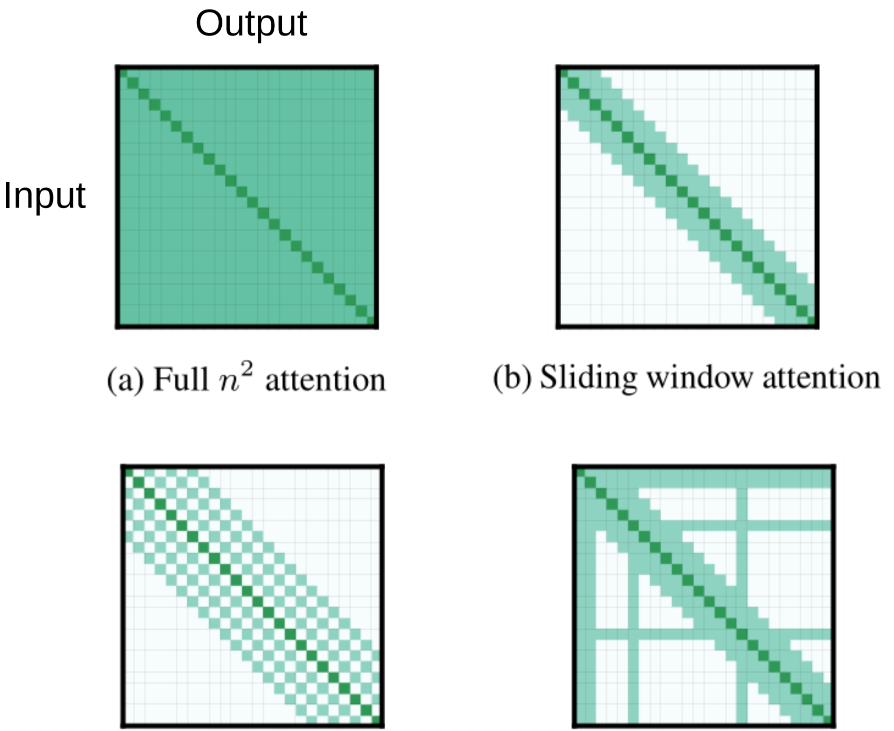
Full attention requires $O(n^2)$

Sliding window requires $O(h \cdot n)$,
h is the window size

Dilated sliding window also requires $O(h \cdot n)$

Global requires $O(g \cdot n)$,
g is the global tokens number

Total: $\mathbf{O((g + h) \cdot n)}$



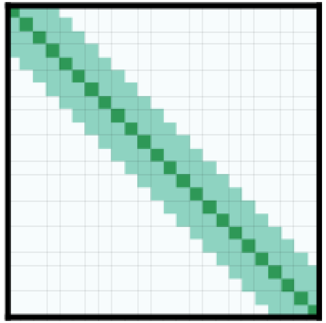
(a) Full n^2 attention (b) Sliding window attention

(c) Dilated sliding window (d) Global+sliding window

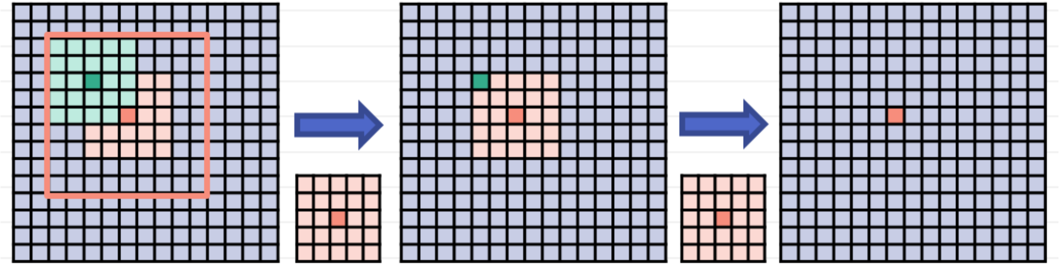
Skoltech

Sliding window attention

Idea is similar to Convolutional Neural Networks – CNNs



(b) Sliding window attention



Receptive field increases along layers for images

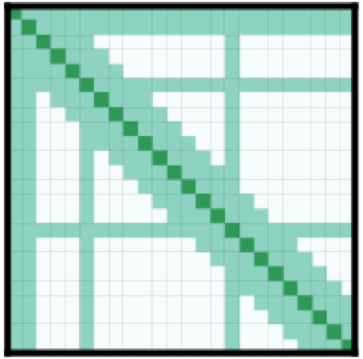
Skoltech

Global tokens

Global requires $O(g \cdot n)$,
g is the global tokens number

Global tokens examples:

- [CLS] is a task-specific token
- [SEP] is a “separator” token



Global+sliding window

Longformer training

General idea:

- start with short sequences and small attention windows,
- increase sequence lengths and attention window sizes at each stage

This idea allows faster training

Number of phases	5
Phase 1 window sizes	32 (bottom layer) - 8,192 (top layer)
Phase 5 window sizes	512 (bottom layer) - (top layer)
Phase 1 sequence length	2,048
Phase 5 sequence length	23,040 (gpu memory limit)
Phase 1 LR	0.00025
Phase 5 LR	000015625
Batch size per phase	32, 32, 16, 16, 16
#Steps per phase (small)	430K, 50k, 50k, 35k, 5k
#Steps per phase (large)	350K, 25k, 10k, 5k, 5k
Warmup	10% of the phase steps with maximum 10K steps
LR scheduler	constant throughout each phase

Hyperparameters

Skoltech

Longformer architecture

Window attention: 512 window size

- we can use a pretrained BERT
- high memory requirements

Global attention:

- question tokens and answer candidates for WikiHop and to question tokens for TriviaQA
- no global attention for the coreference resolution
- [CLS] token for classification problems

Skoltech

BigBird

Skoltech

BigBird attentions

Memory requirements,
n is the sequence length:

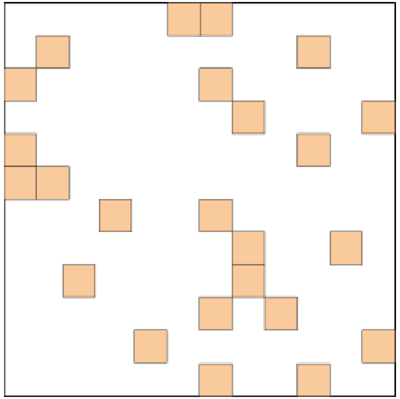
Random attention requires $O(r \cdot n)$

Sliding window requires $O(h \cdot n)$,
h is the window size

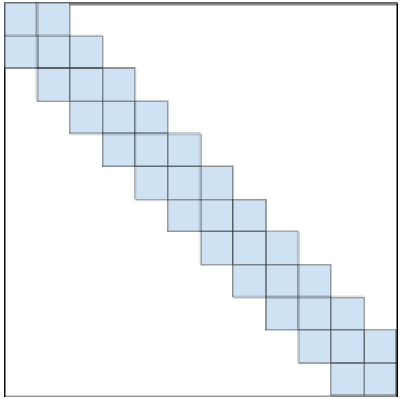
Global requires $O(g \cdot n)$,
g is the global tokens number

BigBird combines <u>3 types of attention mechanism</u>. All of them have linear complexity.

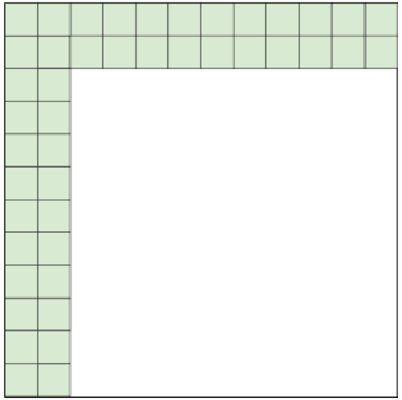
Total: $\mathbf{O((r + h + g) \cdot n)}$



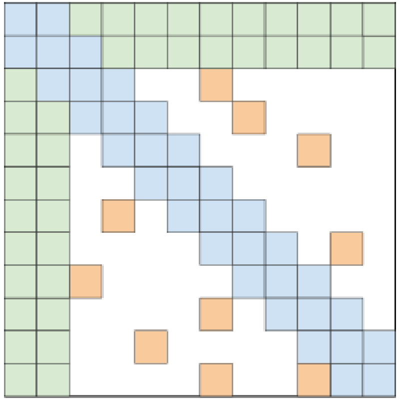
(a) Random attention



(b) Window attention



(c) Global Attention



(d) BIGBIRD

Theoretical results for BigBird

- Universal approximation

Theorem 1. *Given $1 < p < \infty$ and $\epsilon > 0$, for any $f \in \mathcal{F}_{CD}$, there exists a transformer with sparse-attention, $g \in \mathcal{T}_D^{H,m,q}$ such that $d_p(f, g) \le \epsilon$ where D is any graph containing star graph S.*

- Turing completeness

We don't need all attentions for this theorem to hold.
Only <u>Global attention.</u>

Limitations:

Proposition 1. *There exists a single layer full self-attention $g \in \mathcal{T}^{H=1,m=2d,q=0}$ that can evaluate Task 1, i.e. $g(u_1, ..., u_n) = [u_{1^*}, \dots, u_{n^*}]$, but for any sparse-attention graph D with $\tilde{O}(n)$ edges (i.e. inner product evaluations), would require $\tilde{\Omega}(n^{1-o(1)})$ layers.*
We give a formal proof of this fact in App. C.

- In worst case we need $O(n)$ layers – so we'll need $O(n^2)$ computations in total :)
- The problem to solve is to find the corresponding furthest vector

Skoltech

Practical results for BigBird

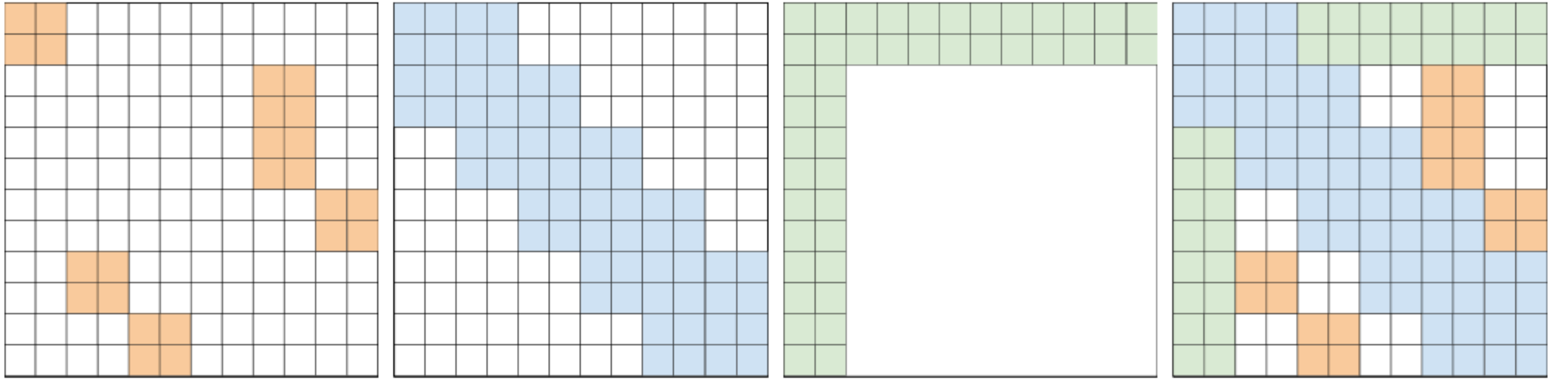
Table 4: Summarization ROUGE score for long documents.

	Model	Arxiv R-1	Arxiv R-2	Arxiv R-L	PubMed R-1	PubMed R-2	PubMed R-L	BigPatent R-1	BigPatent R-2	BigPatent R-L
Prior Art	SumBasic [68]	29.47	6.95	26.30	37.15	11.36	33.43	27.44	7.08	23.66
	LexRank [25]	33.85	10.73	28.99	39.19	13.89	34.59	35.57	10.47	29.03
	LSA [98]	29.91	7.42	25.67	33.89	9.93	29.70	-	-	-
	Attn-Seq2Seq [86]	29.30	6.00	25.56	31.55	8.52	27.38	28.74	7.87	24.66
	Pntr-Gen-Seq2Seq [77]	32.06	9.04	25.16	35.86	10.22	29.69	33.14	11.63	28.55
	Long-Doc-Seq2Seq [20]	35.80	11.05	31.80	38.93	15.37	35.21	-	-	-
	Sent-CLF [82]	34.01	8.71	30.41	45.01	19.91	41.16	36.20	10.99	31.83
	Sent-PTR [82]	42.32	15.63	38.06	43.30	17.92	39.47	34.21	10.78	30.07
	Extr-Abst-TLM [82]	41.62	14.69	38.03	42.13	16.27	39.21	38.65	12.31	34.09
	Dancer [31]	42.70	16.54	38.44	44.09	17.69	40.27	-	-	-
Base	Transformer	28.52	6.70	25.58	31.71	8.32	29.42	39.66	20.94	31.20
	+ RoBERTa [76]	31.98	8.13	29.53	35.77	13.85	33.32	41.11	22.10	32.58
	+ Pegasus [108]	34.81	10.16	30.14	39.98	15.15	35.89	43.55	20.43	31.80
	BigBird-RoBERTa	41.22	16.43	36.96	43.70	19.32	39.99	55.69	37.27	45.56
Large	Pegasus (Reported) [108]	44.21	16.95	38.83	45.97	20.15	41.34	52.29	33.08	41.75
	Pegasus (Re-eval)	43.85	16.83	39.17	44.53	19.30	40.70	52.25	33.04	41.80
	BigBird-Pegasus	**46.63**	**19.02**	**41.77**	**46.32**	**20.65**	**42.33**	**60.64**	**42.46**	**50.01**

Skoltech

Implementation details

Block computations improve utilization of GPUs

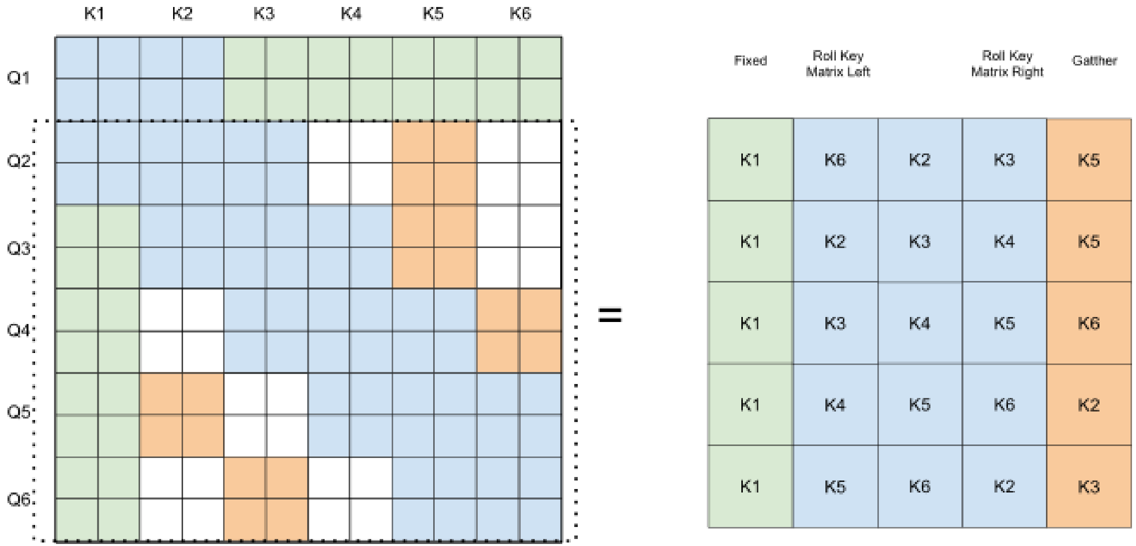


(a) Random Attention (b) Window Attention (c) Global Attention (d) BigBird

Skoltech

Implementation details

Reshaping block computations improves utilization of GPUs
Some edge effects don’t affect quality much



BigBird hyperparameters

The architecture has <u>many global tokens and large window sizes</u>

Parameter	BigBird-ITC	BigBird-ETC
Block length, b	64	84
# of global token, g	$2 \times b$	256
Window length, w	$3 \times b$	$3 \times b$
# of random token, r	$3 \times b$	0
Max. sequence length	4096	4096
# of heads	12	12
# of hidden layers	12	12
Hidden layer size	768	768
Batch size	256	256
Loss	MLM	MLM
Activation layer	gelu	gelu
Dropout prob	0.1	0.1
Attention dropout prob	0.1	0.1
Optimizer	Adam	Adam
Learning rate	10^{-4}	10^{-4}
Compute resources	8×8 TPUv3	8×8 TPUv3

Hyperparameteres for two used BigBird models

Parameter	HotpotQA		NaturalQ		TriviaQA		WikiHop	
Global token location	ITC	ETC	ITC	ETC	ITC	ETC	ITC	ETC
# of global token, g	128	256	128	230	128	320	128	430
Window length, w	192	252	192	252	192	252	192	252
# of random token, r	192	0	192	0	192	0	192	0
Max. sequence length	4096	4096	4096	4096	4096	4096	4096	4096
# of heads	12	12	12	12	12	12	12	12
# of hidden layers	12	12	12	12	12	12	12	12
Hidden layer size	768	768	768	768	768	768	768	768
Batch size	32	32	128	128	32	32	64	64
Loss	cross-entropy golden spans		cross-entropy golden spans		cross-entropy noisy spans [19]		cross-entropy ans choices	
Compute resources	4×2 TPUv3		4×8 TPUv3		4×2 TPUv3		4×4 TPUv3	

Actual hyperparameteres values for QA datasets

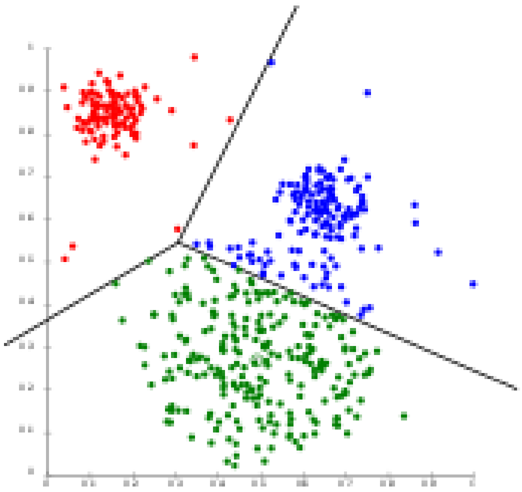
Skoltech

Reformer

Reformer idea

In softmax we mostly pay attention to tokens with the highest scores

Idea:
- Split the whole sequence of tokens to buckets
- Each bucket consists of similar sequences



How to split to buckets?

Model Type	Memory Complexity	Time Complexity
Transformer	$\max(bld_{ff}, bn_h l^2)n_l$	$(bld_{ff} + bn_h l^2)n_l$
Reversible Transformer	$\max(bld_{ff}, bn_h l^2)$	$(bn_h ld_{ff} + bn_h l^2)n_l$
Chunked Reversible Transformer	$\max(bld_{model}, bn_h l^2)$	$(bn_h ld_{ff} + bn_h l^2)n_l$
LSH Transformer	$\max(bld_{ff}, bn_h ln_r c)n_l$	$(bld_{ff} + bn_h n_r lc)n_l$
Reformer	$\max(bld_{model}, bn_h ln_r c)$	$(bld_{ff} + bn_h n_r lc)n_l$

Skoltech

Locality-sensitive hashing: splitting to chunks

Hashing function on representations of tokens: $x \to h(x)$

If x_1, x_2 are *similar*, their hashes $h(x_1), h(x_2)$ are *close to each other*

If x_1, x_2 are *distinct*, their hashes $h(x_1), h(x_2)$ are *far away from each other*

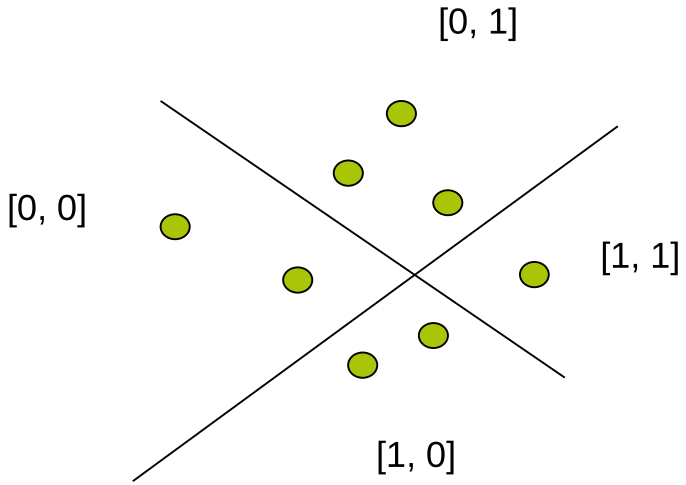
We allow attention only from objects of a single hash backet

Attention Type	Memory Complexity	Time Complexity
Scaled Dot-Product	$\max(bn_h ld_k, bn_h l^2)$	$\max(bn_h ld_k, bn_h l^2)$
Memory-Efficient	$\max(bn_h ld_k, bn_h l^2)$	$\max(bn_h ld_k, bn_h l^2)$
LSH Attention	$\max(bn_h ld_k, bn_h ln_r(4l/n_c)^2)$	$\max(bn_h ld_k, bn_h n_r l(4l/n_c)^2)$

Skoltech

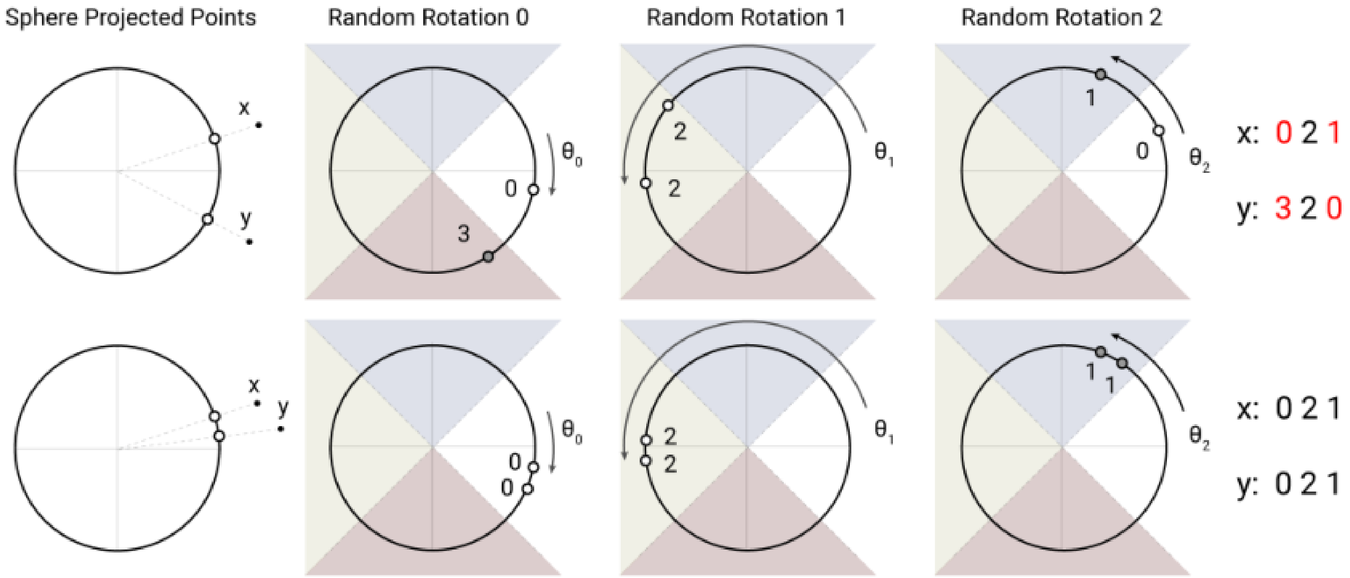
Locality-sensitive hashing: random hash function

Random projection using a number of hyperplanes



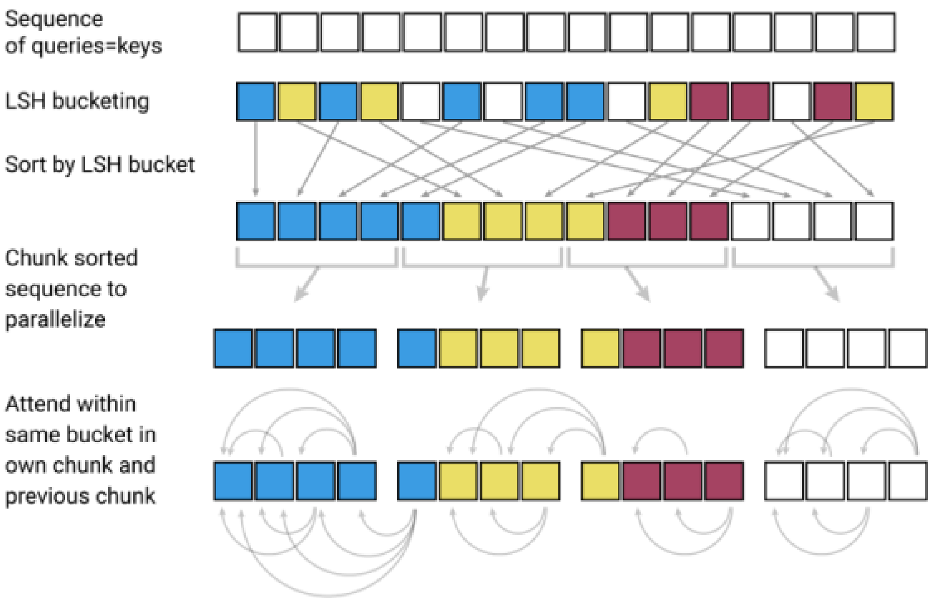
Skoltech

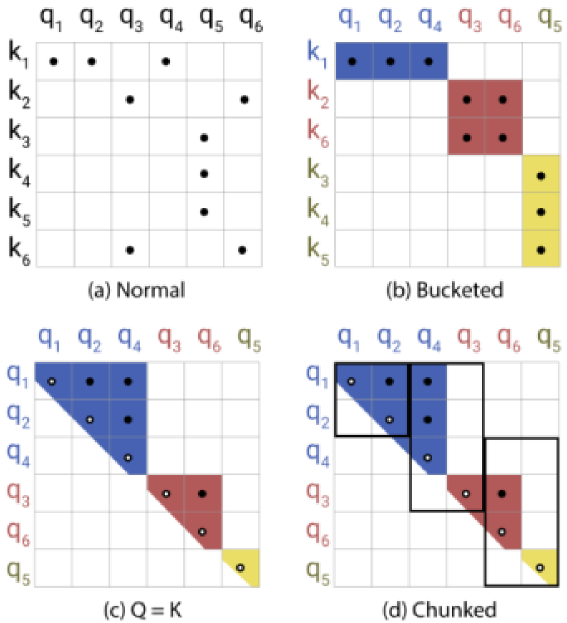
Splitting to chunks: hypersphere view



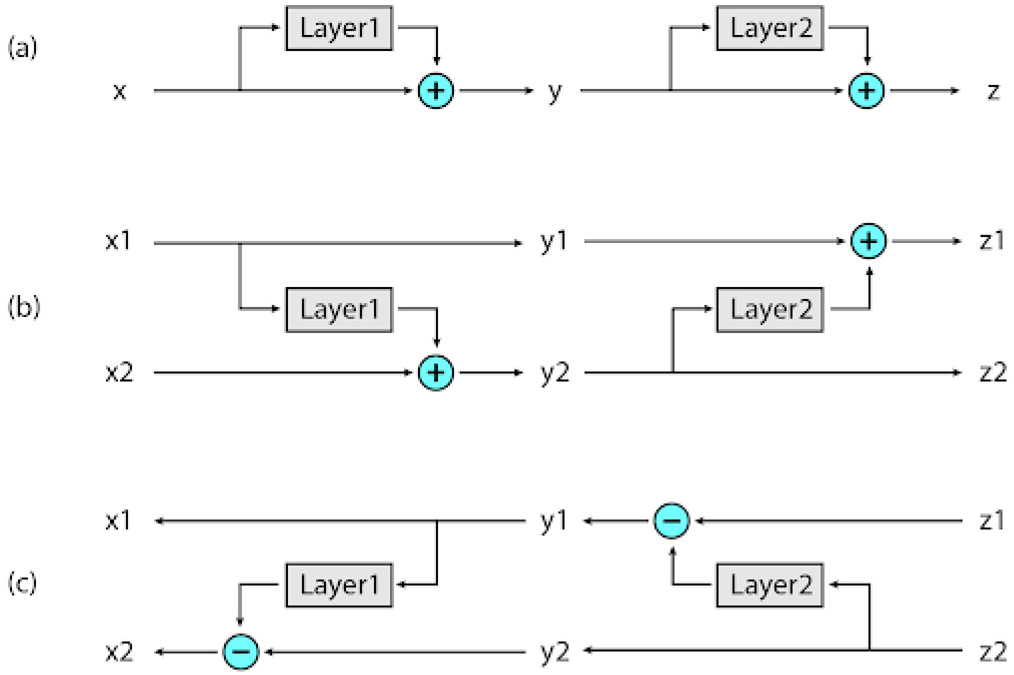
Skoltech

Uniform splitting to chunks



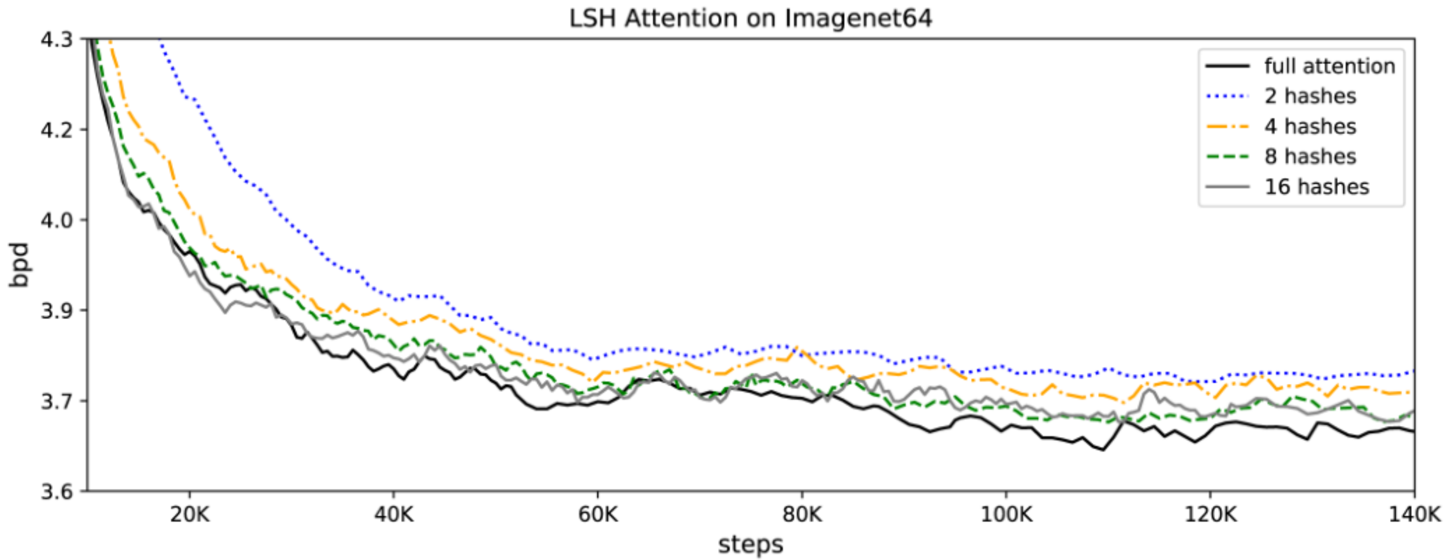


Reform backpropagation



Skoltech

Reform results: ImageNet



Input part

Reform
output

Skoltech

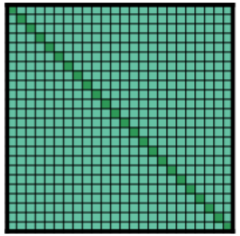
Other ways

Learnable Patterns

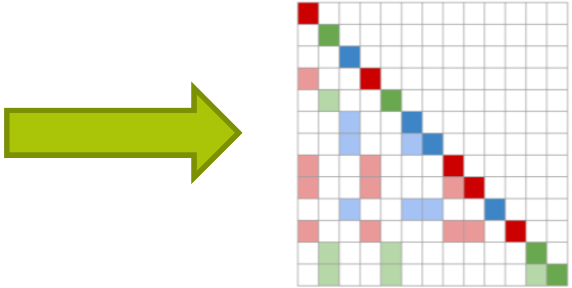
An extension to fixed, pre-determined pattern is learnable ones. Unsurprisingly, models using learnable patterns aim to learn the access pattern in a data-driven fashion. A key characteristic of learning patterns is to determine a notion of token relevance and then assign tokens to buckets or clusters

Conclusion:

- Harder to implement efficiently
- Slower than fixed pattern
- Data-driven fashion could help to find optimal attention matrix pattern

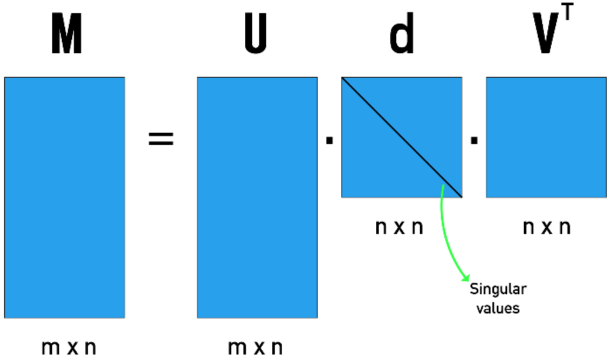


(a) Full n^2 attention



Low-Rank Methods

Another emerging technique is to improve efficiency by leveraging low-rank approximations of the self-attention matrix. The key idea is to assume a low-rank structure in the N × N matrix.



Conclusion:

- Could not directly compute low-rank approximation

Linformer

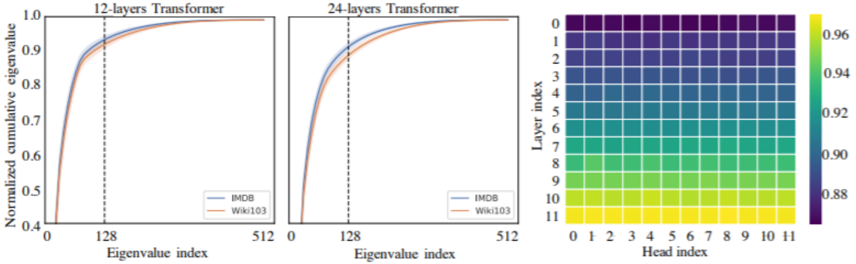
Perform an SVD decomposition in each self-attention matrix, which adds additional complexity. Projects the length dimension of keys and values to a lower-dimensional representation (N → k).

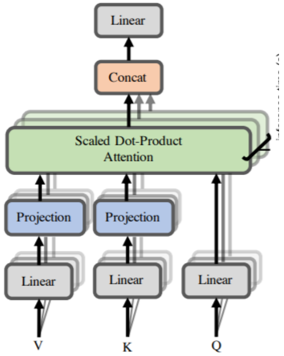
An optimal value of k dimension does not depend on a length of sequence.

Complexity:

$$O(n),$$

Where n – length of sequence





Transformers for long sequences

Model / Paper	Complexity	Decode	Class
Memory Compressed† (Liu et al., 2018)	$\mathcal{O}(n_c^2)$	✓	FP+M
Image Transformer† (Parmar et al., 2018)	$\mathcal{O}(n.m)$	✓	FP
Set Transformer† (Lee et al., 2019)	$\mathcal{O}(nk)$	✗	M
Transformer-XL† (Dai et al., 2019)	$\mathcal{O}(n^2)$	✓	RC
Sparse Transformer (Child et al., 2019)	$\mathcal{O}(n\sqrt{n})$	✓	FP
Reformer† (Kitaev et al., 2020)	$\mathcal{O}(n \log n)$	✓	LP
Routing Transformer (Roy et al., 2020)	$\mathcal{O}(n \log n)$	✓	LP
Axial Transformer (Ho et al., 2019)	$\mathcal{O}(n\sqrt{n})$	✓	FP
Compressive Transformer† (Rae et al., 2020)	$\mathcal{O}(n^2)$	✓	RC
Sinkhorn Transformer† (Tay et al., 2020b)	$\mathcal{O}(b^2)$	✓	LP
Longformer (Beltagy et al., 2020)	$\mathcal{O}(n(k+m))$	✓	FP+M
ETC (Ainslie et al., 2020)	$\mathcal{O}(n_g^2 + nn_g)$	✗	FP+M
Synthesizer (Tay et al., 2020a)	$\mathcal{O}(n^2)$	✓	LR+LP
Performer (Choromanski et al., 2020)	$\mathcal{O}(n)$	✓	KR
Linformer (Wang et al., 2020b)	$\mathcal{O}(n)$	✗	LR
Linear Transformers† (Katharopoulos et al., 2020)	$\mathcal{O}(n)$	✓	KR
Big Bird (Zaheer et al., 2020)	$\mathcal{O}(n)$	✗	FP+M

FP = Fixed Patterns, LP = Learnable Pattern, LR = Low Rank,
KR = Kernel, RC = Recurrence, M = Memory.

Long Range Arena: A Benchmark for Efficient Transformers

Tay, Yi, et al. "Long range arena: A benchmark for efficient transformers." *arXiv preprint arXiv:2011.04006* (2020).

Long-Range Arena (LRA) benchmark (pronounced el-ra)

A core focus of LRA is assessing how different Xformers capture long-range dependencies.

Benchmark includes the task, evaluators, and models in Python 3 and Jax/Flax. Code is open-sourced.

- https://github.com/google/flax
- https://github.com/google-research/long-range-arena

Desiderata

- Generality (include tasks that only require encoding)
- Simplicity (avoid including any particular data augmentation and pretraining)
- Challenging
- Long inputs
- Probing diverse aspects (ability to model relations and hierarchical/spatial structures, generalization capability)
- Non-resource intensive and accessible

Tasks

- Long listops (10-way classification task, 2K length)

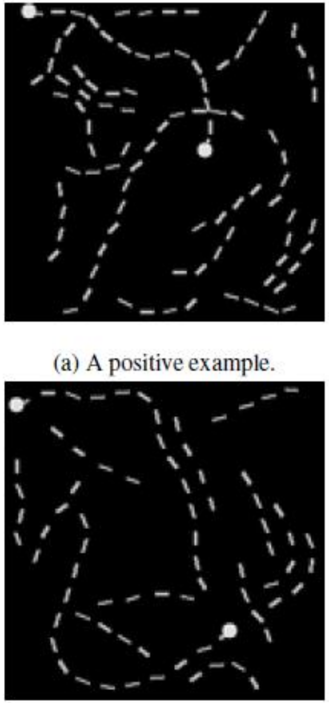
```
INPUT: [MAX 4 3 [MIN 2 3 ] 1 0 [MEDIAN 1 5 8 9, 2]]      OUTPUT: 5
```

- Byte-level text classification (IMDB dataset, 50K movie reviews with label positive/negative, binary classification task, 4K length)
- Byte-level document retrieval (ACL Anthology Network Dataset, which identifies if two papers have a citation link, binary classification task, 8K length)
- Image classification on sequences of pixels (CIFAR-10 dataset, grayscale image 32x32 -> sequence 1024)

Tasks

- Pathfinder (image 32x32 -> sequence 1024, whether two points connected with dash line or not, binary classification task)

- Pathfinder-X (Pathfinder for extreme lengths, images128x128 -> sequence 16K)



(a) A positive example.

(b) A negative example.

Figure 1: Samples of the Pathfinder task.

Required attention span

- A trained attention-based model and a sequence of tokens are inputs,
- The required attention span of an attention module is the mean distance between *the query token* and *the attended tokens*, scaled by attention weights.
- The mean required attention span is computed over all attention modules in a vanilla Transformer model for each task, averaged over 1K samples from the validation.

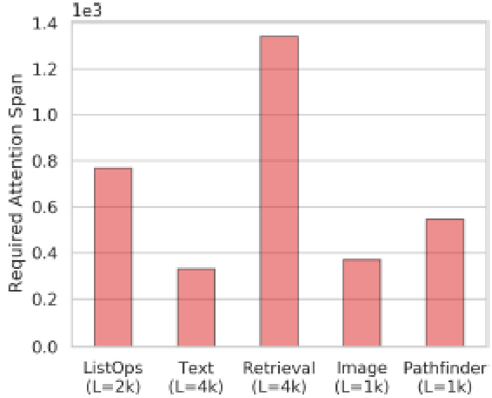


Figure 2: Required attention span on different tasks.

Skoltech

Results

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Transformer	36.37	64.27	57.46	42.44	71.40	FAIL	<u>54.39</u>
Local Attention	15.82	52.98	53.39	41.46	66.63	FAIL	46.06
Sparse Trans.	17.07	63.58	**59.59**	**44.24**	71.71	FAIL	51.24
Longformer	35.63	62.85	56.89	42.22	69.71	FAIL	53.46
Linformer	35.70	53.94	52.27	38.56	<u>76.34</u>	FAIL	51.36
Reformer	**37.27**	56.10	53.40	38.07	68.50	FAIL	50.67
Sinkhorn Trans.	33.67	61.20	53.83	41.23	67.45	FAIL	51.39
Synthesizer	<u>36.99</u>	61.68	54.67	41.61	69.45	FAIL	52.88
BigBird	36.05	64.02	<u>59.29</u>	40.83	74.87	FAIL	**55.01**
Linear Trans.	16.13	**65.90**	53.09	42.34	75.30	FAIL	50.55
Performer	18.01	<u>65.40</u>	53.82	<u>42.77</u>	**77.05**	FAIL	51.41
Task Avg (Std)	29 (9.7)	61 (4.6)	55 (2.6)	41 (1.8)	72 (3.7)	FAIL	52 (2.4)

Table 1: Experimental results on Long-Range Arena benchmark. Best model is in boldface and second best is underlined. All models do not learn anything on Path-X task, contrary to the Pathfinder task and this is denoted by FAIL. This shows that increasing the sequence length can cause seriously difficulties for model training. We leave Path-X on this benchmark for future challengers but do not include it on the Average score as it has no impact on relative performance.

Time and memory usage of Xformers

Model	Steps per second 1K	2K	3K	4K	Peak Memory Usage (GB) 1K	2K	3K	4K
Transformer	8.1	4.9	2.3	1.4	0.85	2.65	5.51	9.48
Local Attention	9.2 (1.1x)	8.4 (1.7x)	7.4 (3.2x)	7.4 (5.3x)	0.42	0.76	1.06	1.37
Linformer	<u>9.3</u> (1.2x)	9.1 (1.9x)	8.5 (3.7x)	7.7 (5.5x)	**0.37**	**0.55**	0.99	**0.99**
Reformer	4.4 (0.5x)	2.2 (0.4x)	1.5 (0.7x)	1.1 (0.8x)	0.48	0.99	1.53	2.28
Sinkhorn Trans	9.1 (1.1x)	7.9 (1.6x)	6.6 (2.9x)	5.3 (3.8x)	0.47	0.83	1.13	1.48
Synthesizer	8.7 (1.1x)	5.7 (1.2x)	6.6 (2.9x)	1.9 (1.4x)	0.65	1.98	4.09	6.99
BigBird	7.4 (0.9x)	3.9 (0.8x)	2.7 (1.2x)	1.5 (1.1x)	0.77	1.49	2.18	2.88
Linear Trans.	9.1 (1.1x)	<u>9.3</u> (1.9x)	<u>8.6</u> (3.7x)	<u>7.8</u> (5.6x)	**0.37**	<u>0.57</u>	**0.80**	<u>1.03</u>
Performer	**9.5** (1.2x)	**9.4** (1.9x)	**8.7** (3.8x)	**8.0** (5.7x)	**0.37**	0.59	<u>0.82</u>	1.06

Table 2: Benchmark results of all Xformer models with a consistent batch size of 32 across all models. We report relative speed increase/decrease in comparison with the vanilla Transformer in brackets besides the steps per second. Memory usage refers to per device memory usage across each TPU device. Benchmarks are run on 4x4 TPU V3 Chips.

Overall results

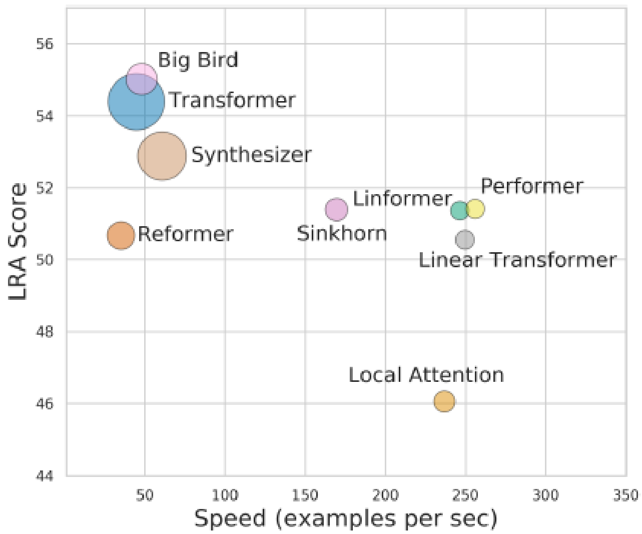


Figure 3: Performance (y axis), speed (x axis), and memory footprint (size of the circles) of different models.

Skoltech

Benchmarking for Efficient Transformers for transactions data

Skoltech

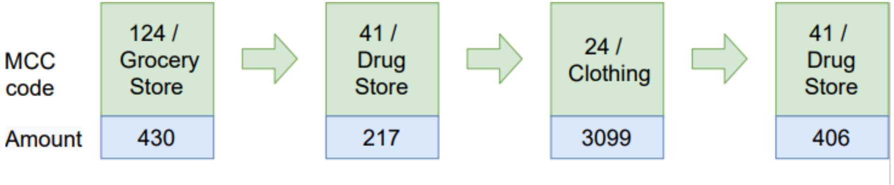
Dataset description
Transaction dataset

Task
Predict the probability of user default

Features
~10 categorical features (MCC code, type etc)
~10 numerical (amount, time etc)

Train test split
Train – 1 million users
Validation – 250k users
Out-of-time – 500k users

Length
700 - mean transaction count
(*Typical transformer length 512)*
Metric
Gini coefficient



52

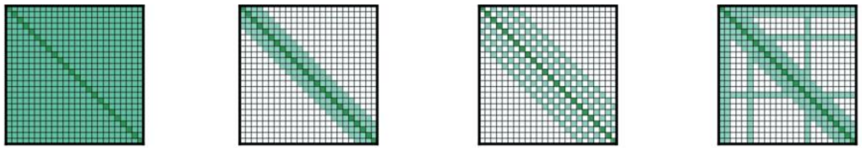
Hyperparameters search for Longformer

Quality
Longformer model shows better quality than baseline LSTM

Hyperparameters
- Very deep model required a huge amount of memory for training
- Sliding window size W does not change quality significant

Baseline metric
LSTM - 60.12 ± 0.17

№	Heads	Embedding size	W	Layers	Hidden dropout	OOT Gini	RAM (Mb)
1	8	512	128	1	0.1	59.59	**7647**
2	8	256	128	1	0.1	58.84	4929
3	8	64	128	1	0.1	60.11	4581
4	8	64	128	2	0.1	61.02	6039
5	8	64	256	2	0.1	61.03	10131
6	8	64	64	2	0.1	60.58	4500
7	4	64	128	8	0.3	62.21	13971
8	4	32	128	8	0.3	62.58	13505
9	4	64	128	8	0.5	62.39	13971
10	4	64	128	12	0.3	62.39	20255
11	2	64	128	8	0.5	62.47	12891
12	4	64	128	12	0.6	**62.83**	20255



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

53

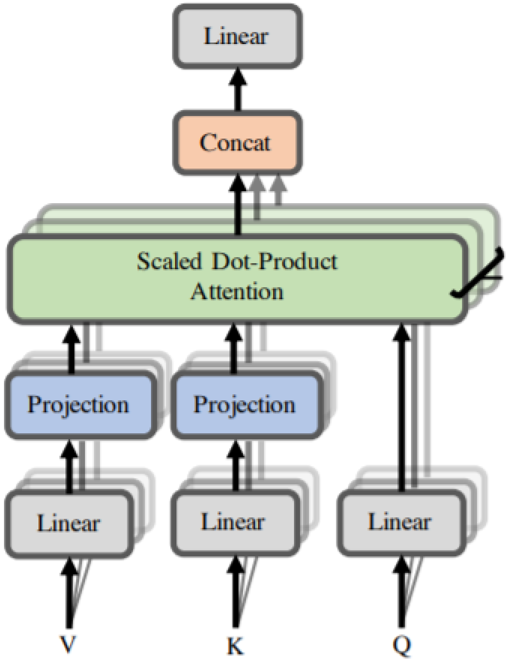
Results of the experiment

Matrix projection Linformer

Quality
Linformer with projection matrix shows poor results in comparing with recurrent neural network models

Shift invariance of a projection matrix
The projection matrix not being shifting invariant

Shift invariance test
We train the model to prove this property, choose a subsample, and calculate the metric on it. Then we shift all transactions by one and calculate the metric again.



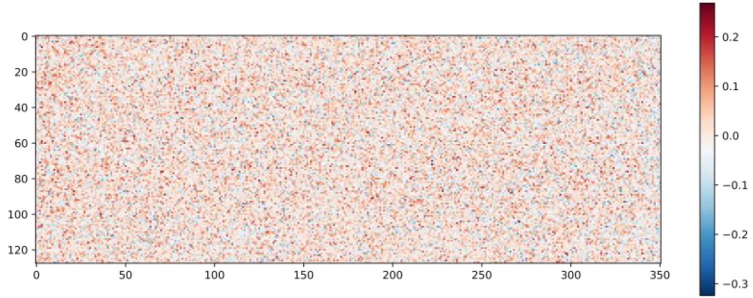
Skoltech

Results of the experiment
Matrix projection Linformer

Quality
Linformer with projection matrix shows poor results in comparing with recurrent neural network models

Shift invariance of a projection matrix
The projection matrix not being shifting invariant

Shift invariance test
We train the model to prove this property, choose a subsample, and calculate the metric on it. Then we shift all transactions by one and calculate the metric again.



Linformer projection matrix

Dataset	Gini
Original dataset	53.47
Shift by one transaction	47.31

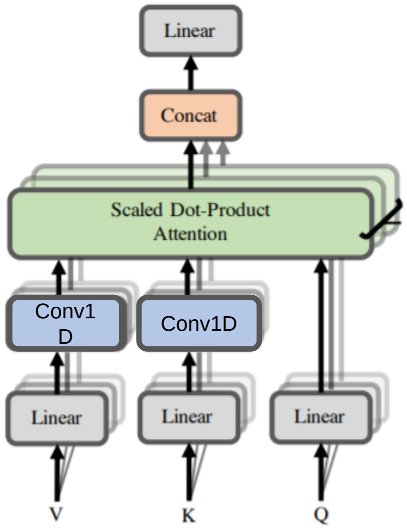
Skoltech

Results of the experiment
Convolution Linformer

Quality
Convolution Linformer shows a better result using less memory and shorter training time.

Convolution modification
To make projection shift-invariant, we replace a projection matrix with a convolution layer.

Cycle-shift augmentation
The result of convolution with kernel and stride equals size could change after adding a new object in sequence. (+1 Gini)



Skoltech

Wang, Sinong, et al. "Linformer: Self-attention with linear complexity." *arXiv preprint arXiv:2006.04768* (2020).

56

Results of the experiment

Convolution Linformer

Metric
LSTM - 60.12±0.17
Longformer - 62.80

Quality
Convolution Linformer shows a better result using less memory and shorter training time.

Convolution modification
To make projection shift-invariant, we replace a projection matrix with a convolution layer.

Cycle-shift augmentation
The result of convolution with kernel and stride equals size could change after adding a new object in sequence. (+1 Gini)

№	Projection size	Heads	Embedding size	Layers	Hidden dropout	dim ff	OOT gini	RAM (Mb)
1	128	8	64	4	0.2	2048	63.5	8757
2	32	8	64	4	0.2	2048	62.02	5124
3	64	8	64	4	0.2	2048	62.40	6789
4	128	4	64	4	0.2	2048	62.73	7159
5	128	8	64	2	0.2	2048	64.04	5333
6	128	8	64	4	0.2	2048	62.49	8715
7	128	8	32	4	0.2	2048	64.17	8179
8	128	8	64	4	0.2	2048	62.98	3635
9	128	8	64	3	0.2	2048	63.53	7051
10	128	8	64	2	0.2	512	**64.3 ± 0.19**	5333
11	128	8	64	2	0.2	256	63.5	**3395**
12	128	8	128	2	0.2	512	63.26	3833

Skoltech

57

Results of the experiment
Convolution Linformer

Curriculum learning: Pre-trained weights
Training first on short sequences and the fine-tune on long show better quality and reduce training time

Non linear complexity
We need to increase the projection size: increasing length of sequence without increasing projection size leads to quality degradation.

№	Length	K	Training type	OOT gini	RAM (Mb)	Train time (hours)
1	350	128	From scratch	64.37	9787	3
2	1500	128	From scratch	63.75	12451	4.5
3	1500	256	From scratch	64.81	16047	11.61
4	1500	256	Pre-trained weights	**65.2**	16081	9.15
5	1500	400	From scratch	65.39	23481	17.3
6	1500	400	Pre-trained/train only conv layer	64.05	2299	7.1
7	1500	400	Pre-trained weights	**66.00 $\pm$ 0.21**	23489	8.5

Skoltech

58

Results of the experiment

Comparison LSTM and Transformer

Transaction count	LSTM	Longformer	Linformer	Convolution Linformer
350	60.12 ± 0.17	62.80	53.47	64.37 ± 0.19
1500	63.01 ± 0.14	None	None	66.00 ± 0.21

Skoltech

59

Conclusions

- We can improve Transformers memory and computational requirements with some heuristics and different types of attention

- The main ideas are:
  - combine different sparse attention mechanisms
  - bucket processing

- But there are some limitations in terms of reducing number of parameters and memory as we have large window sizes and large number of global attention nodes

Skoltech