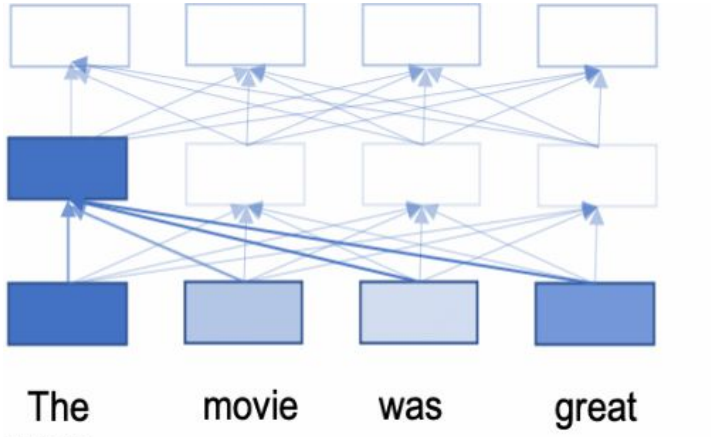
Transformer Models

Transformer Models

- Based on **Self-attention**
- Representation of each input location is a linear combination of other input elements in previous layer
  - Weighted by learned "attention" values



UWNLP AI2

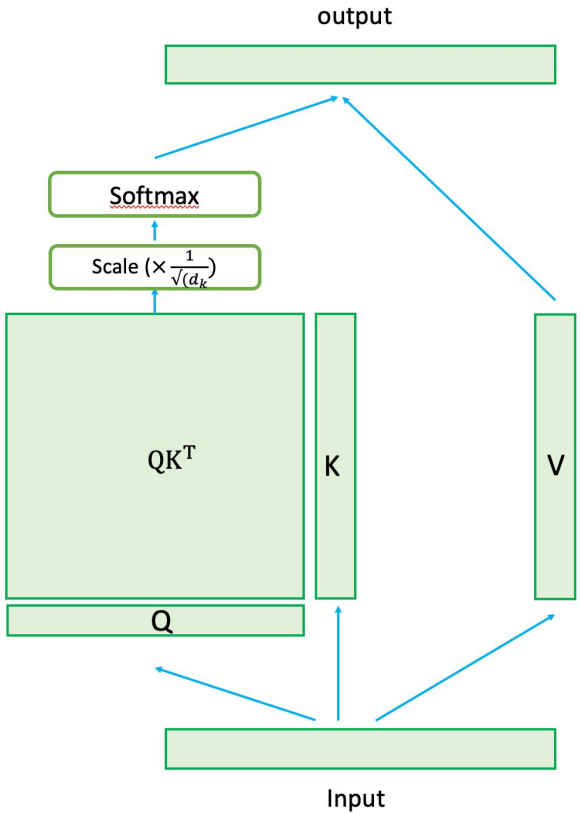
Transformer attention

- Scaled dot-product attention

$$\text{Attention}: \mathbb{R}^{\text{key}} \times \mathbb{R}^{\text{seq}\times\text{key}} \times \mathbb{R}^{\text{seq}\times\text{val}} \to \mathbb{R}^{\text{val}}$$

$$\text{Attention}(Q, K, V) = \underset{\text{seq}}{\text{softmax}}\left(\frac{Q \underset{\text{key}}{\odot} K}{\sqrt{|\text{key}|}}\right) \underset{\text{seq}}{\odot} V.$$

- **Q**, **K**, and **V** are all derived from the input **X**
- Attention weights: Dot product attention weights between Q and K
- Final contextualized representation:
  Linear combination of values **V** using the attention weights



Transformer

Add a non-linear feed forward layer after attention layer

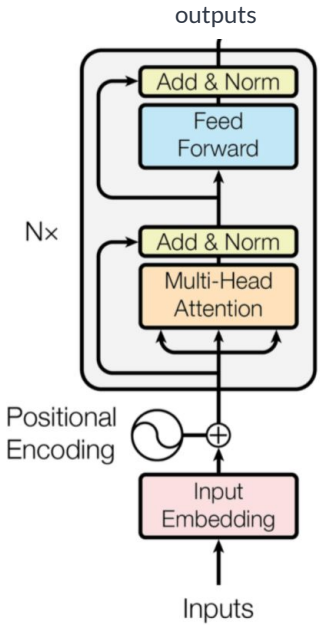
Positions?

Learned position embeddings added to the input
Can be *Absolute* or *Relative* (Shaw et al, 2018)

Multi-headed:

Each head can conceptually focus on different parts of the input

Can parallelize computation, unlike RNNs



UWNLP AI2

Transformer, encoder only

Encoder only (e.g., BERT)

One stack of transformer layers

Self-attention

Computation cost: $O(N^2)$, N: sequence length

Transformer encoder-decoder

Encoder-Decoder
Machine translation

h = Encoder(English)
French = Decoder(***h***)

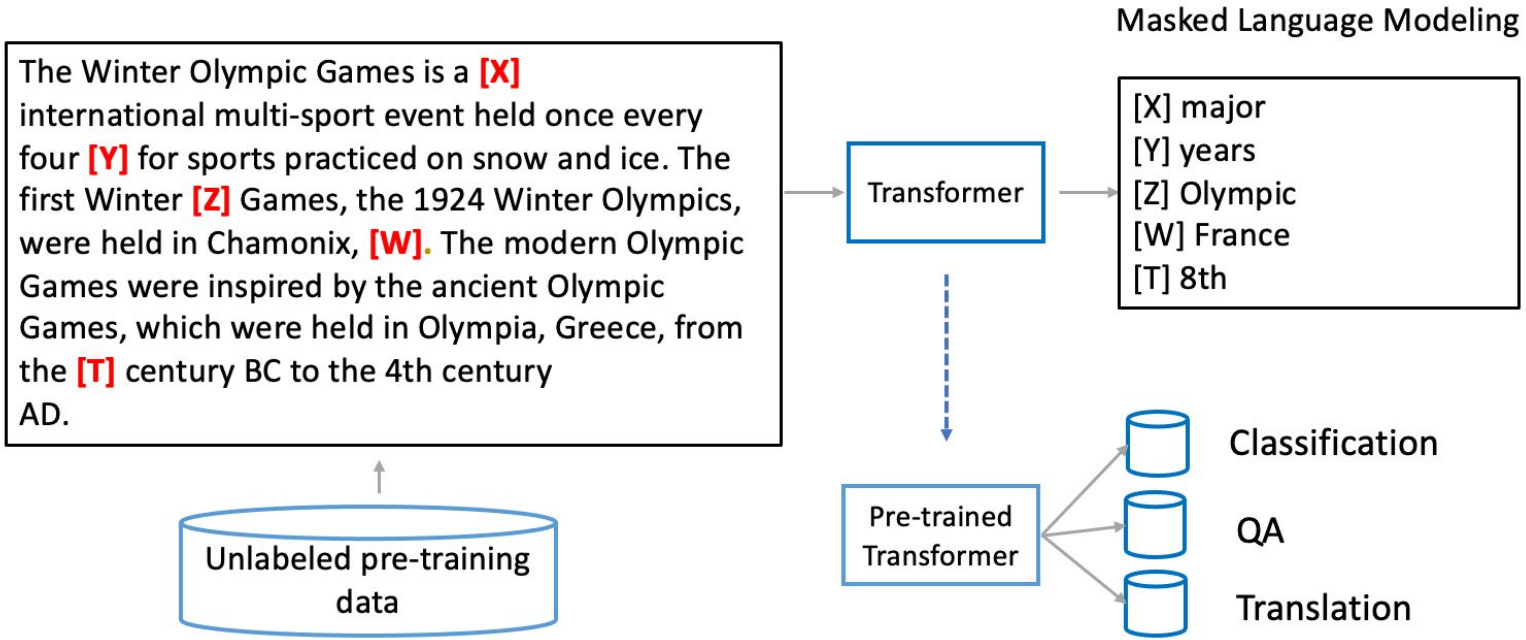
N = Enc. sequence length
M = Dec. sequence length

Enc. Attention = $O(N^2)$
Dec. Attention = $O(N*M + M^2)$

Fig source: https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

UWNLP AI2

Pre-trained Transformers

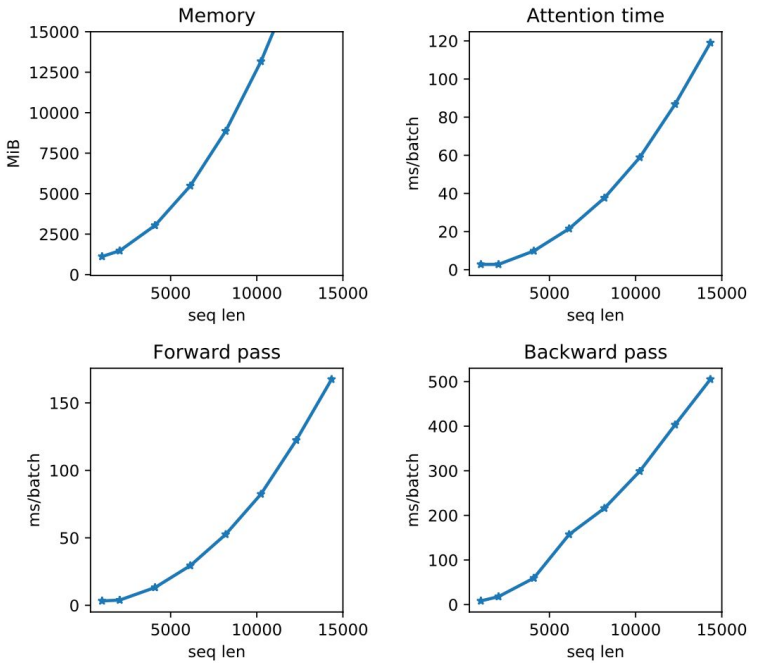


Longer inputs?

Recall the computational cost of self-attention O(N^2)

Quickly becomes a bottleneck for long sequences

Pre-trained Transformers have limited input capacity
(e.g., BERT 512 tokens)



Single-head single-layer transformer
Titan RTX8000 GPU

UWNLP

AI2

Extending Transformers to longer inputs

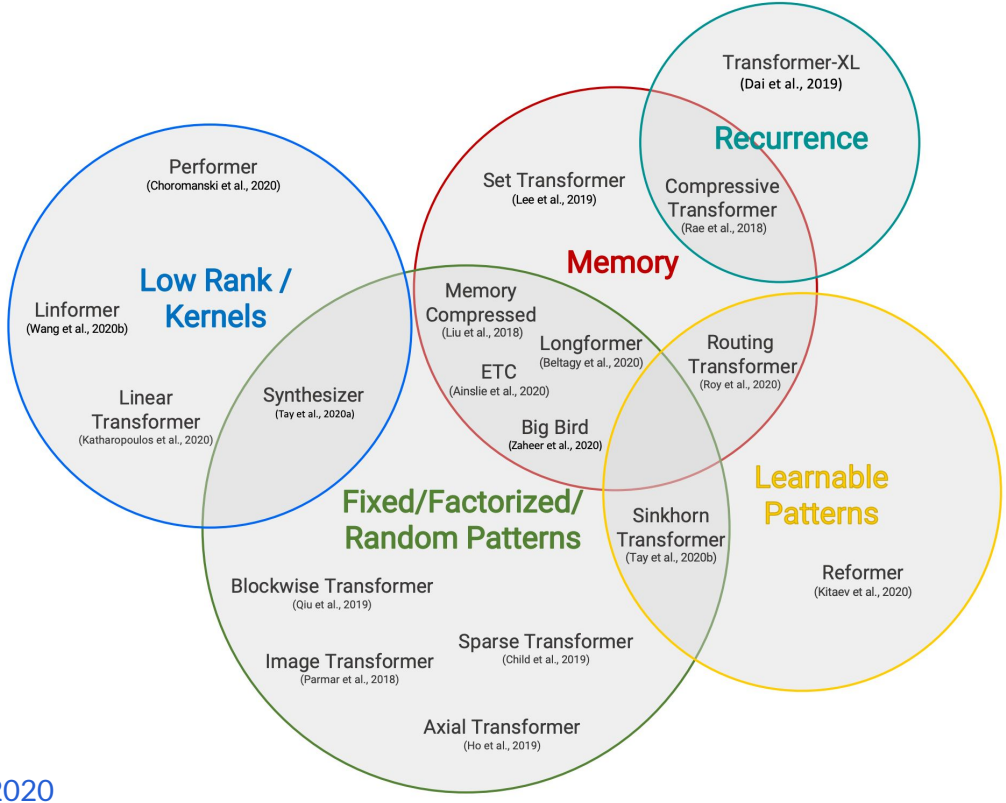


Fig reference: [Tay et al., 2020]

Extending Transformers to longer inputs

A wide variety of recently proposed method to make Transformers efficient for large sequences

We will cover important works in each category and key ideas

We will not be able to cover all existing methods

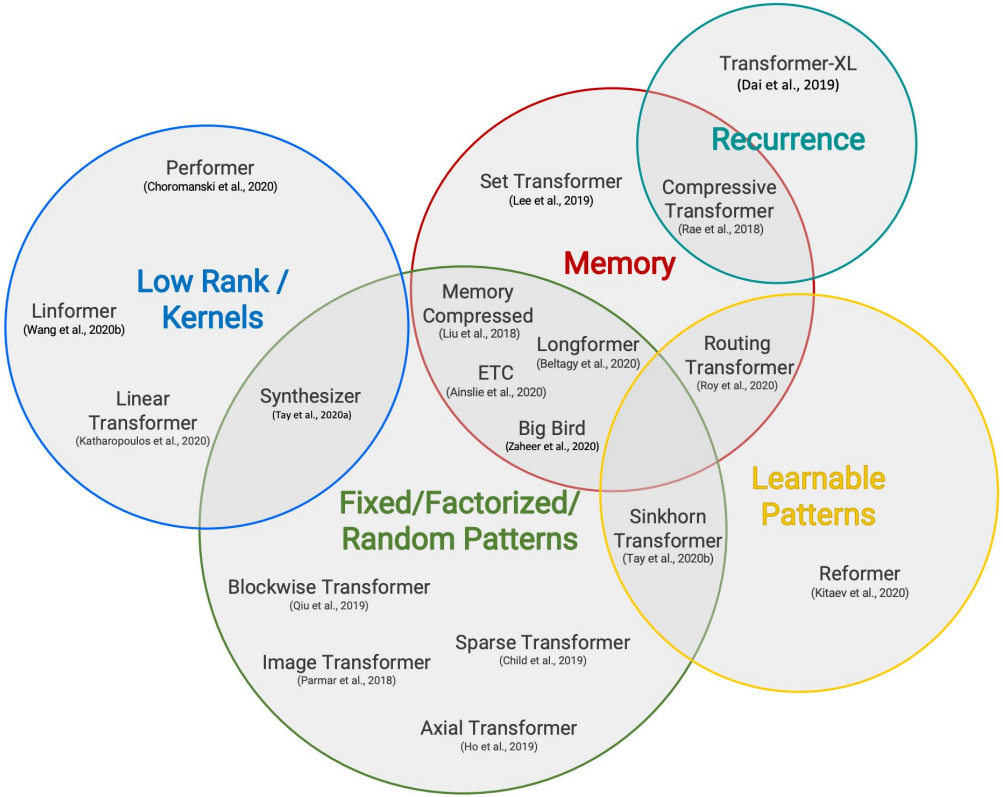


Fig reference: [Tay et al., 2020]

1 - Recurrence

Recurrence and compressing memory

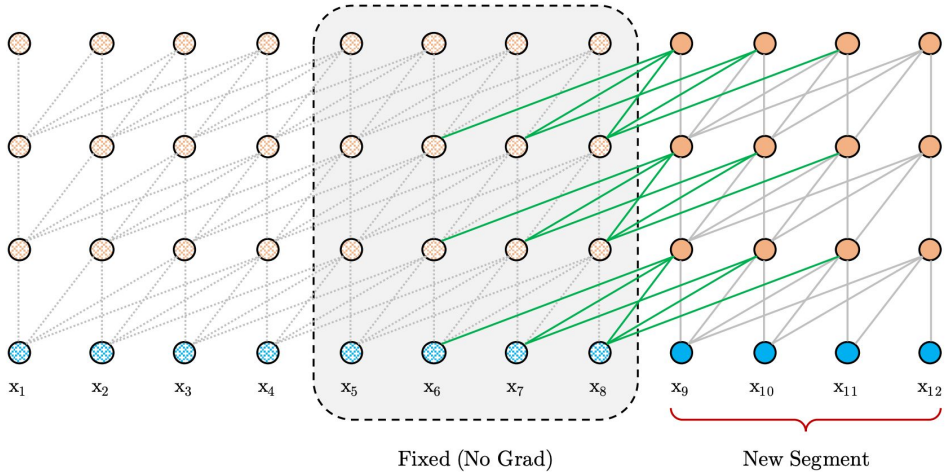
Transformer-XL (Dai et al., 2019)

Segment-level recurrence

Representations from previous steps are cached and reused

No gradient updates on previous segments

Use of relative position embeddings in attention computation

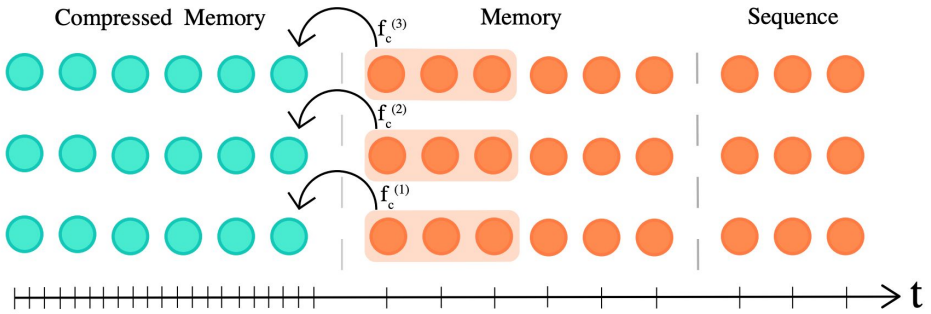


Recurrence and compressing memory

Compressive Tranformer (Rae et al., 2019)

Similar to Transformer-XL

- Instead of discarding old past activations it compresses them
- Keeping two types of memory
  - A primary memory
  - An additional compressed memory

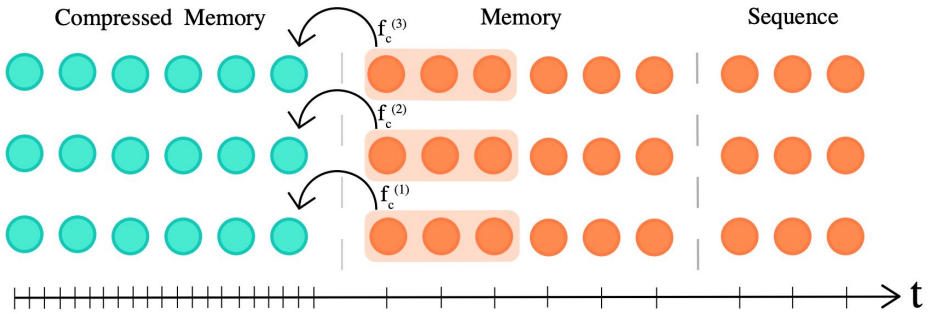


Recurrence and compressing memory

Compressive Tranformers (Rae et al., 2019)

Approaches to compress the memory:

1. mean/max pooling
2. 1D convolutions
3. Dilated convolutions
4. Most used (e.g., sorted by usage of attention).



An additional auto-encoding loss

learns to reconstruct the original memory from its compressed version

UWNLP AI2

Recurrence and compressing memory

Compressive Tranformers (Rae et al., 2019)

Table 4: State-of-the-art results on Enwik8.

Model	BPC
7L LSTM (Graves, 2013)	1.67
LN HyperNetworks Ha et al. (2016)	1.34
LN HM-LSTM Chung et al. (2016)	1.32
ByteNet (Kalchbrenner et al., 2016)	1.31
RHN Zilly et al. (2017)	1.27
mLSTM Krause et al. (2016)	1.24
64L Transf. Al-Rfou et al. (2019)	1.06
24L TXL (Dai et al., 2019)	0.99
Sparse Transf. (Child et al., 2019)	0.991
Adaptive Transf. (Sukhbaatar et al., 2019)	0.98
24L TXL (ours)	0.98
24L Compressive Transformer	**0.97**

Table 5: Compression approaches on Enwik8.

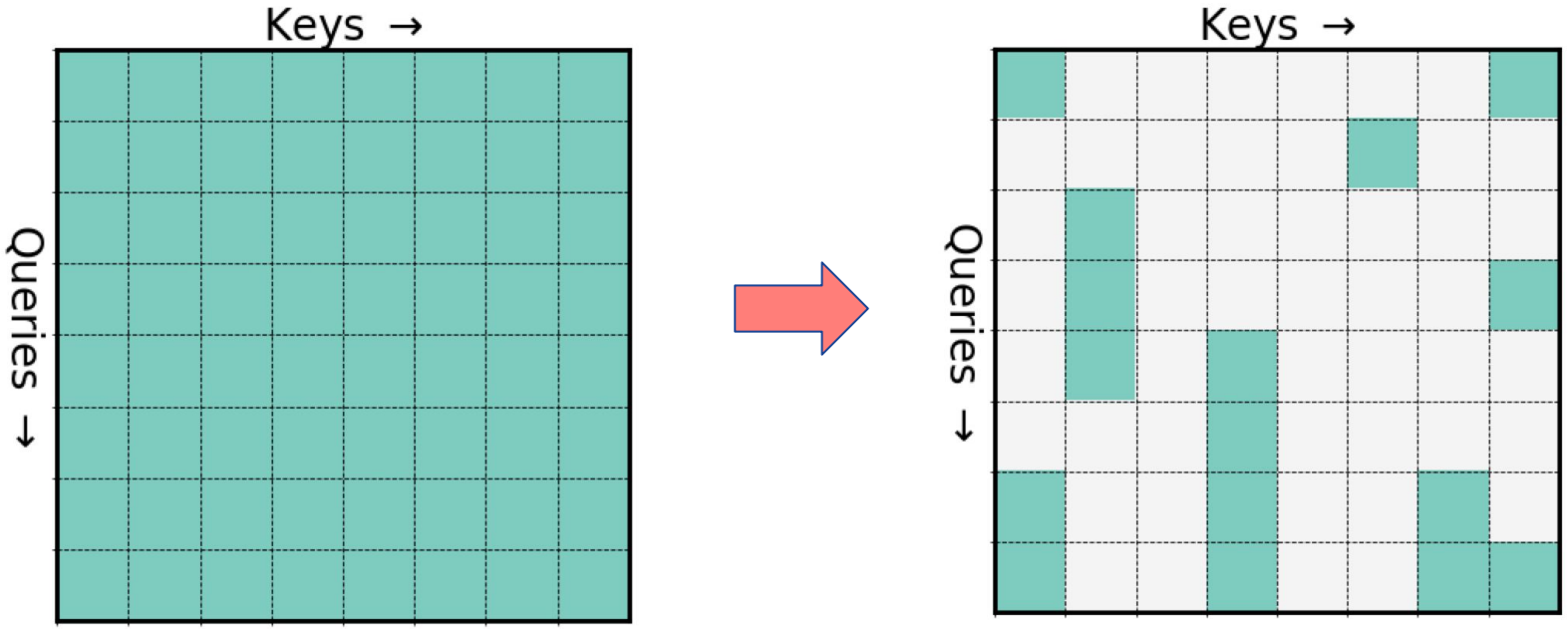
Compression fn	Compression loss	BPC
Conv	BPTT	0.996
Max Pooling	N/A	0.986
Conv	Auto-encoding	0.984
Mean Pooling	N/A	0.982
Most-used	N/A	0.980
Dilated conv	Attention	0.977
Conv	Attention	**0.973**

UWNLP AI2

2- Sparse patterns

Sparse patterns

Key idea is to make the attention operation sparse

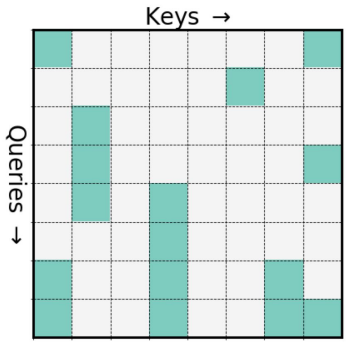


UWNLP AI2

Sparse patterns

How to sparsify?

- Content-based
  - Routing Transformers (Roy et al., 2019)
  - Sinkhorn (Tay et al., 2020)
  - Reformer (Kitaev et al., 2020)
- Pre-specified patterns
  - Sparse Transformer (Child et al., 2019)
  - Longformer (Beltagy, et al., 2020)
  - ETC (Ainslie et al., 2020)
  - BigBird (Zaheer et al., 2020)



UWNLP AI2

Content based patterns

Reformer (Kitaev et al., 2020)

Using Locality Sensitive Hashing (LSH)

Efficient & approximate way of nearest neighbors search in high dimensional datasets

Select *hash* functions such that if *x* is close to *y* then with high enough probability we have *hash(x) == hash(y)*

Find LSH hashes of *Q* and *K* matrices.

Compute attention only for the *k* and *q* vectors within the same hash buckets.

Content based patterns- Reformer (Kitaev et al., 2020)

LSH attention

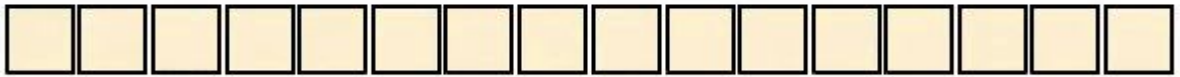
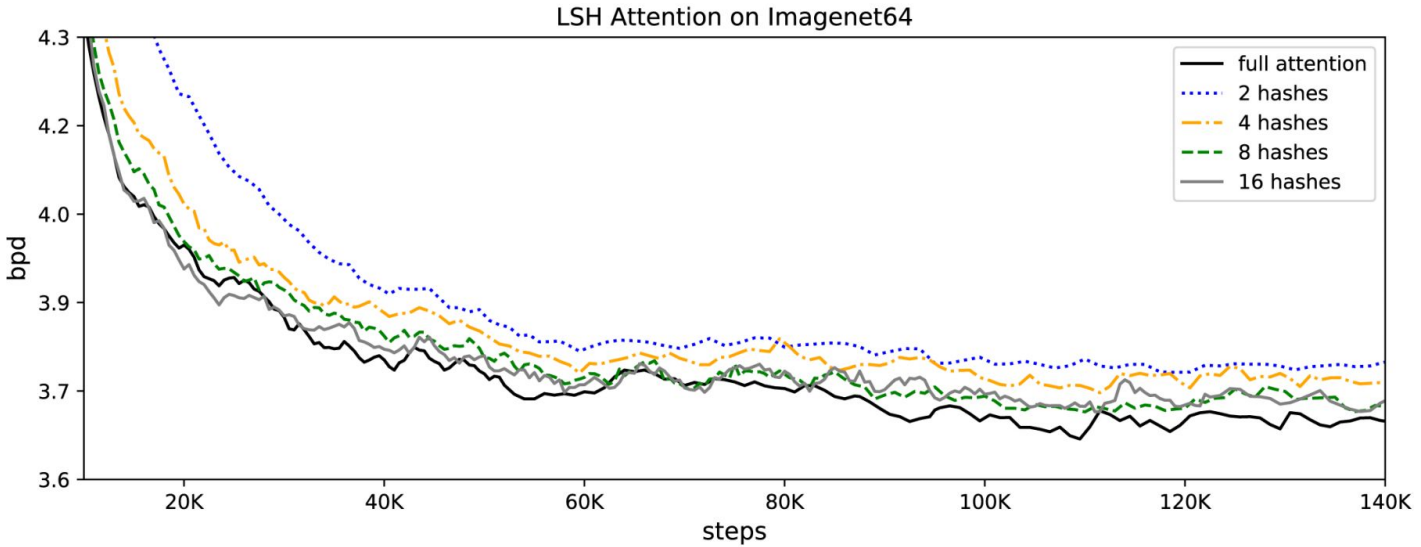


Figure from illustrating reformer blog

Content based patterns - Reformer (Kitaev et al., 2020)

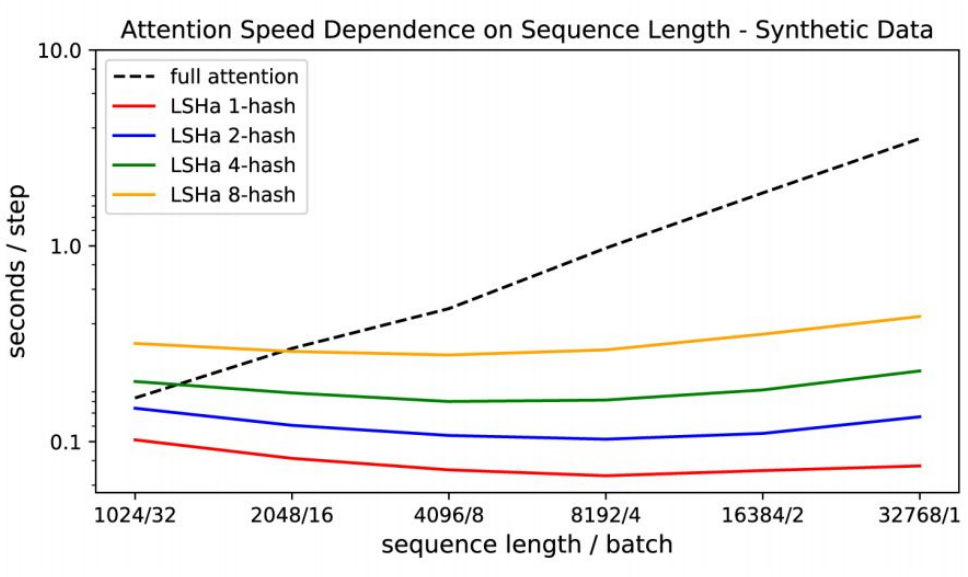
Model quality



UWNLP AI2

Content based patterns - Reformer (Kitaev et al., 2020)

Performance speed



UWNLP AI2

Content based patterns - Reformer (Kitaev et al., 2020)

Additional contribution: Using Reversible Resnets (Gomez et al., 2017)

Allow the activations at any given layer to be recovered from the activations at the following layer, using only the model parameters.

Normal residual connection $y = x + F(x)$

Reversible: works on a pairs of inputs/outputs

$$y_1 = x_1 + F(x_2) \qquad y_2 = x_2 + G(y_1)$$

A layer can be reversed by subtracting (rather than adding) the residuals

$$x_2 = y_2 - G(y_1) \qquad x_1 = y_1 - F(x_2)$$

Content based patterns - Reformer (Kitaev et al., 2020)

Reversible layers

$$y_1 = x_1 + F(x_2) \qquad y_2 = x_2 + G(y_1)$$

$$x_2 = y_2 - G(y_1) \qquad x_1 = y_1 - F(x_2)$$

F: Attention, G: Feed Forward

$$Y_1 = X_1 + \text{Attention}(X_2) \qquad Y_2 = X_2 + \text{FeedForward}(Y_1)$$

UWNLP AI2

Content based patterns - Reformer (Kitaev et al., 2020)

Not storing activations doesn't hurt performance at all.

Model	BLEU	*sacreBLEU* *Uncased*[3]	*sacreBLEU* *Cased*[4]
Vaswani et al. (2017), base model	27.3		
Vaswani et al. (2017), big	28.4		
Ott et al. (2018), big	29.3		
Reversible Transformer (base, 100K steps)	27.6	*27.4*	*26.9*
Reversible Transformer (base, 500K steps, no weight sharing)	28.0	*27.9*	*27.4*
Reversible Transformer (big, 300K steps, no weight sharing)	29.1	*28.9*	*28.4*

UWNLP AI2

Content based patterns

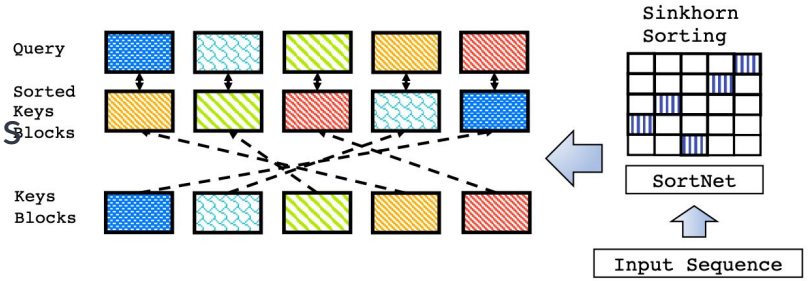
Sparse Sinkhorn Attention (Tay et al., 2020)

Key idea: Use block-wise local attention but sort the key blocks

Differentiable sorting of internal representations within self-attention

Using a differentiable Sinkhorn balancing mechanism

After sorting, attention computation is efficient even in local neighborhood



Content based patterns

Routing Transformer (Roy et al., 2020)

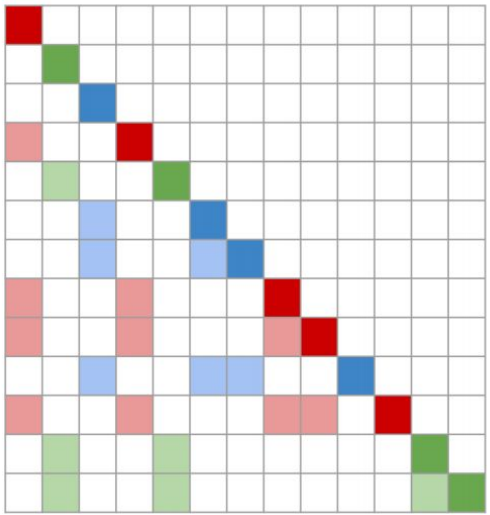
Clustering-based attention mechanism that learns the attention sparsity in a data driven fashion.

Project Q and K into routing matrix R

$$R = QW_R + KW_R$$

The R matrix undergoes k-means clustering

After clustering, each token only attends to tokens from the same cluster



Auto-regressive attention, each color shows cluster membership

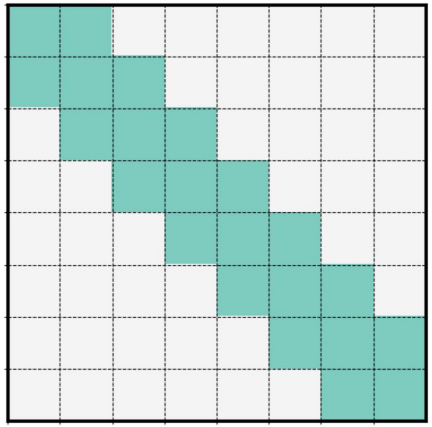
UWNLP AI2

Sparsifying the attention

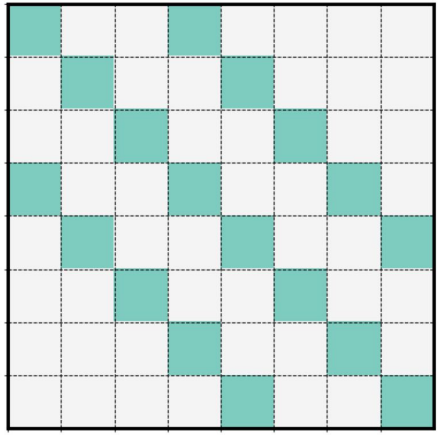
Longformer (Beltagy et al., 2020)

Sparse Transformer (Child et al., 2019) - no global attention

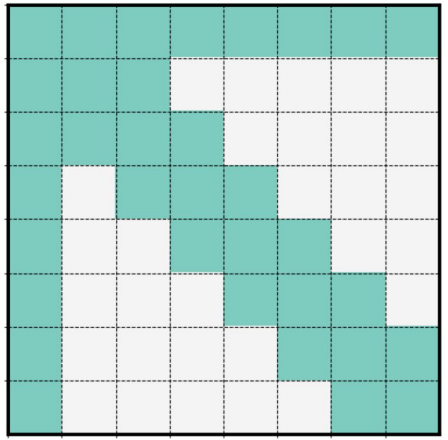
Local attention



Dilated attention



Local + global attention



UWNLP

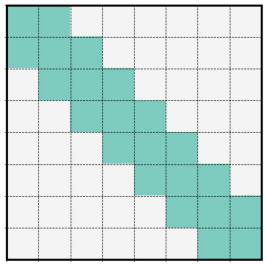
AI2

Pre-specified sparsity patterns

Sparsify the attention matrix (QK^T)

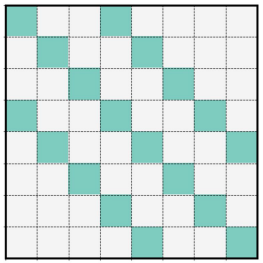
A variety of patterns has been explored in the past work

Sliding window



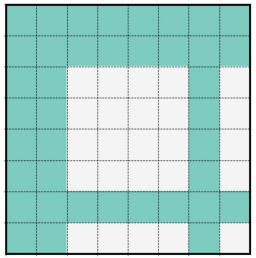
Sparse Transformer
Longformer

Dilated



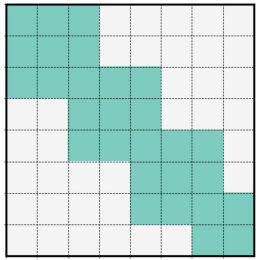
Longformer

Global



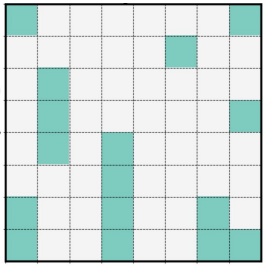
Longformer
ETC
Big Bird
GMAT

Blocked



Block-wise
ETC
Big Bird
Sinkhorn

Random



Big Bird

UWNLP AI2

Pre-specified sparsity patterns

Why sparsifying the attention works?

- Important theoretical results from **Big Bird** (Zaheer et al., 2020)
  - Sparse Transformers are Universal Approximators of Seq2Seq functions
  - Sparse encoder-decoder transformers are Turing Complete
  - Caveat: Need more layers
    - A task that can be solved in O(1) layers under standard complexity theoretic assumptions, requires Ω(~n) layers for any sparse attention layers with O˜(n) edges.

UWNLP AI2

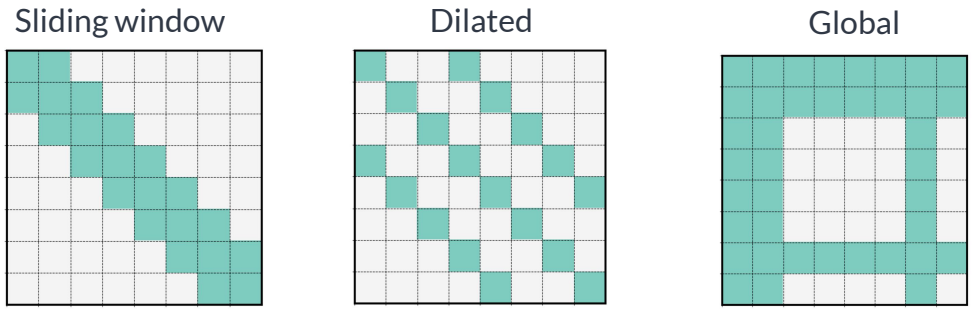
Sparsifying attention

Sparse patterns also used in GPT-3 (Brown et al., 2020)

- Alternating locally banded sparse attention and dense attention between layers

Pre-specified sparsity patterns

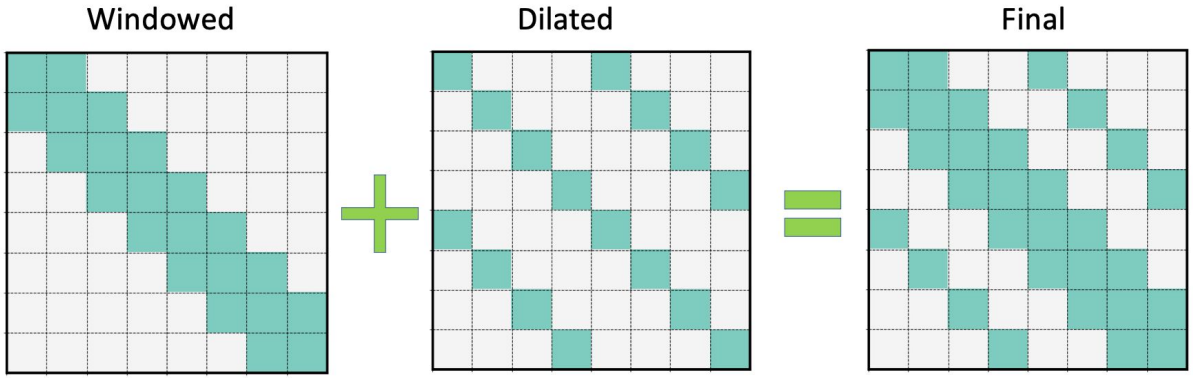
Type of patterns, **Longformer** (Beltagy, et al., 2020)



Pre-specified sparsity patterns

Type of patterns, **Longformer** (Beltagy, et al., 2020)

Language modeling (no global attention pattern)

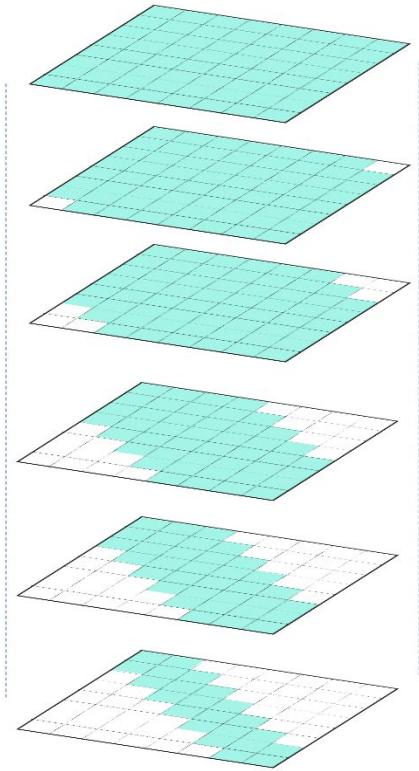


Pre-specified sparsity patterns

Even with windowed attention receptive field can be large with multiple layers

- Information can flow across the entire input

- Further increase of receptive field with **dilation**
  - Longformer (Beltagy et al., 2020)
- Different layers and attention heads can follow different patterns
  - Longformer (Beltagy et al., 2020)



Pre-specified sparsity patterns

Longformer (Beltagy et al., 2020) - Char LM results

Model	#Param	Dev	Test
Dataset `text8`			
T12 (Al-Rfou et al., 2018)	44M	-	1.18
Adaptive (Sukhbaatar et al., 2019)	38M	1.05	1.11
BP-Transformer (Ye et al., 2019)	39M	-	1.11
Our Longformer	41M	1.04	**1.10**
Dataset `enwik8`			
T12 (Al-Rfou et al., 2018)	44M	-	1.11
Transformer-XL (Dai et al., 2019)	41M	-	1.06
Reformer (Kitaev et al., 2020)	-	-	1.05
Adaptive (Sukhbaatar et al., 2019)	39M	1.04	1.02
BP-Transformer (Ye et al., 2019)	38M	-	1.02
Our Longformer	41M	1.02	**1.00**

Table 2: *Small* model BPC on `text8` & `enwik8`

Model	#Param	Test BPC
Transformer-XL (18 layers)	88M	1.03
Sparse (Child et al., 2019)	≈100M	0.99
Transformer-XL (24 layers)	277M	0.99
Adaptive (Sukhbaatar et al., 2019)	209M	0.98
Compressive (Rae et al., 2020)	277M	0.97
Routing (Roy et al., 2020)	≈223M	0.99
Our Longformer	102M	0.99

Table 3: Performance of *large* models on `enwik8`

Model	Dev BPC
No Dilation	1.21
Dilation on 2 heads	**1.20**

Local + Dilation patterns

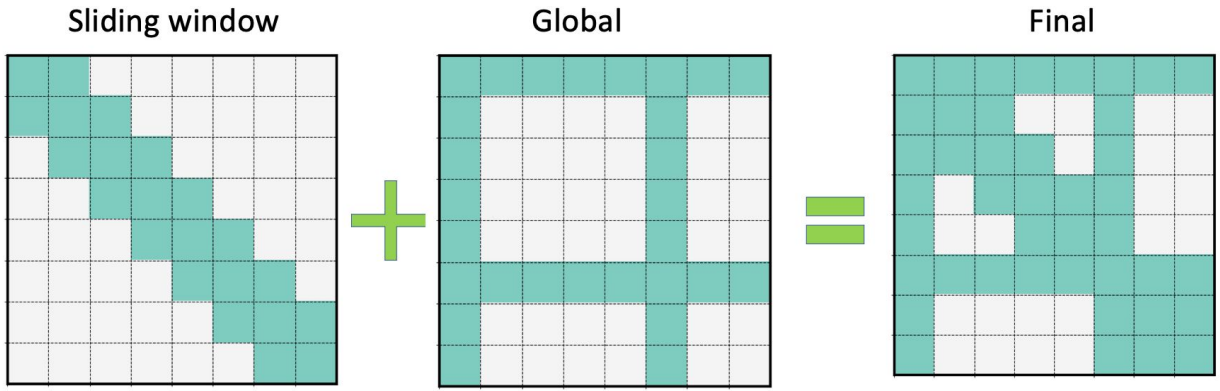
UWNLP AI2

Pre-specified sparsity patterns

What patterns to use? **Longformer** (Beltagy, et al., 2020)

For downstream NLP tasks

- Global pattern depends on the task (important tokens for the task).
- No dilation



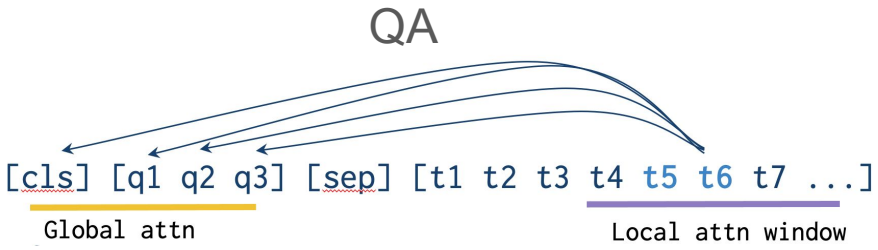
UWNLP AI2

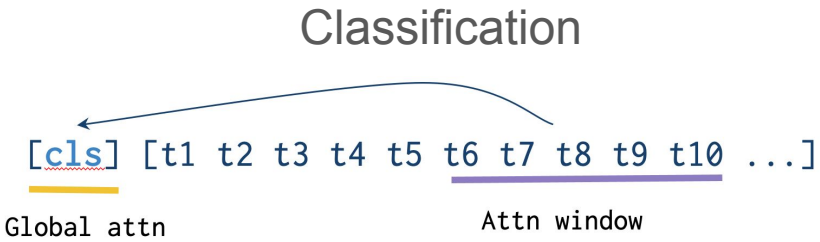
Pre-specified sparsity patterns

Longformer (Beltagy, et al., 2020)

Global attention is task specific

- Indirect information flow is not always sufficient.
- For some tasks direct attention is needed.

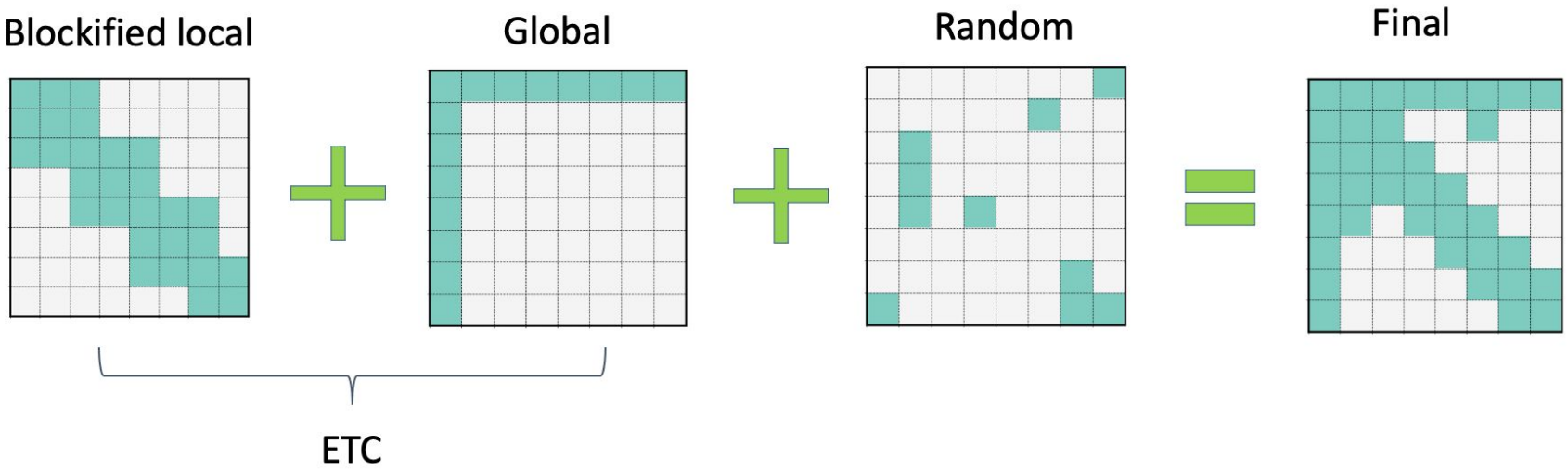




UWNLP AI2

Pre-specified sparsity patterns

What patterns to use? **Big Bird** (Zaheer et al., 2020) and **ETC** (Ainslie et al., 2020)



UWNLP AI2

Pre-specified sparsity patterns

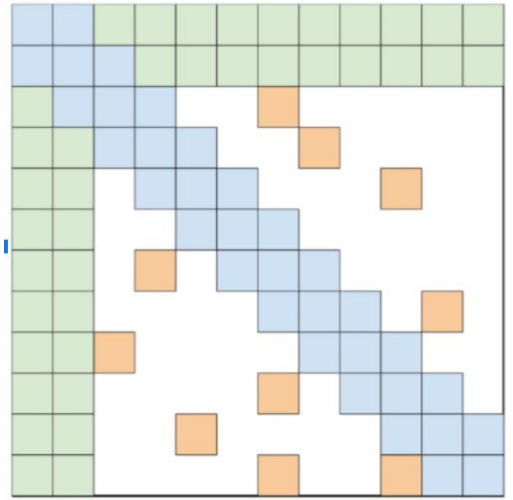
Big Bird (Zaheer et al., 2020) and **ETC** (Ainslie et al., 2020)

Special "global" tokens are used in the beginning of the sequence

Serving as "global memory"

- Also in **GMAT** (Gupta et al., 2020)

Global attention is used on these tokens



Assign 1 global tokens per sentence

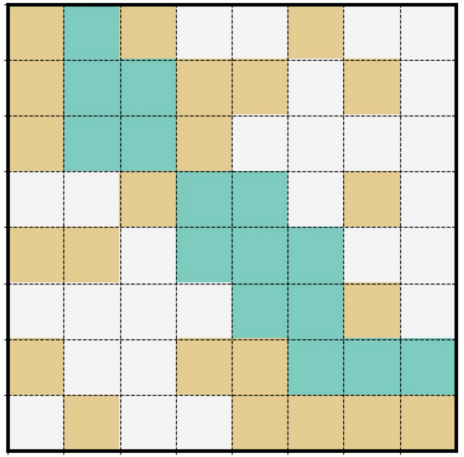
Use relative position labels and masking to have each global token access its corresponding sentence

UWNLP AI2

Sparsifying attention

Big Bird (Zaheer et al., 2020)

Local and Random sparse attention not enough to match performance of full attention



Model	MLM	SQuAD	MNLI
BERT-base	64.2	88.5	83.4
Random (R)	60.1	83.0	80.2
Window (W)	58.3	76.4	73.1
R + W	62.7	85.1	80.5

Table 1: Building block comparison @512

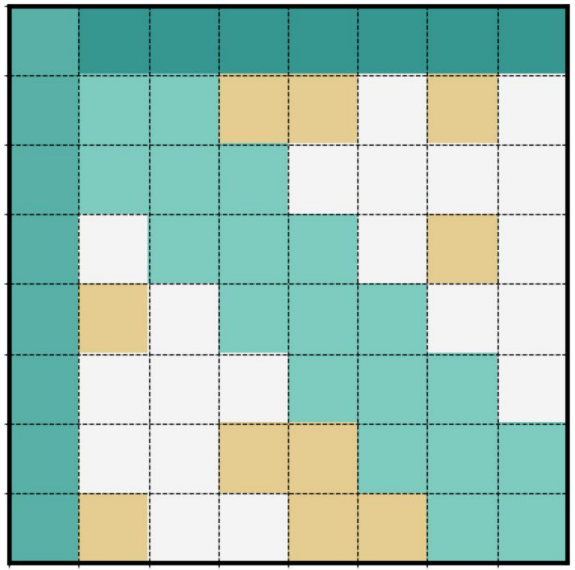
UWNLP AI2

Pre-specified sparsity patterns

Big Bird (Zaheer et al., 2020)

Global attention is important

Model	MLM	SQuAD	MNLI
BERT-base	64.2	88.5	83.4
Random (R)	60.1	83.0	80.2
Window (W)	58.3	76.4	73.1
R + W	62.7	85.1	80.5
Global + R + W	64.4	87.2	82.9



UWNLP AI2

Pre-specified sparsity patterns

Longformer vs Big Bird / ETC

- Longformer global attention is more versatile (can be applied to any token)
- Longformer also uses dilation (only for LM)
- Big Bird / ETC uses relative positions embeddings to capture structure
- Big Bird / ETC uses more pre-training as well as a second CPC loss

Pre-specified sparsity patterns

Ok sparsity is great, but how to efficiently implement this?

Challenge:

Arbitrary sparse matrix multiplication is not supported in DL libraries

Pre-specified sparsity patterns

Challenge: Efficient implementation

Solutions:

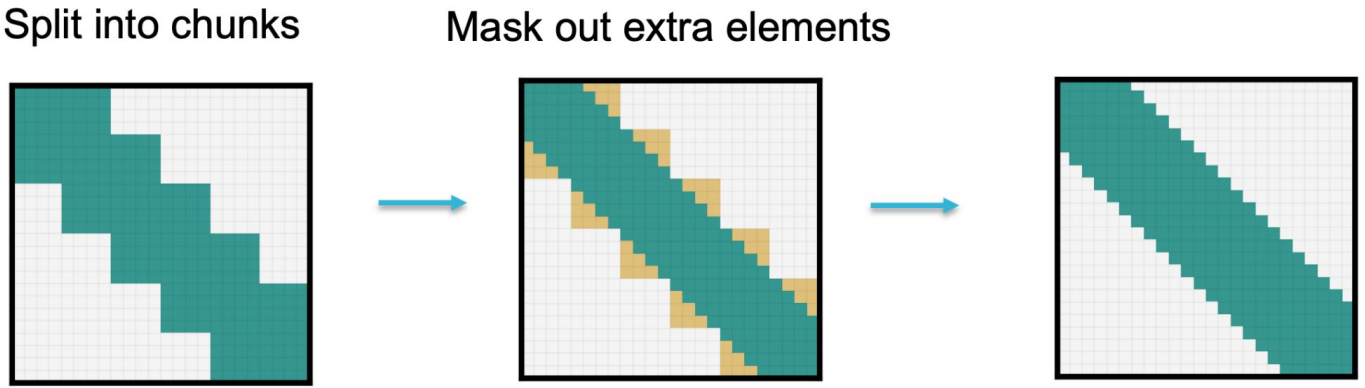
- Perform computations in blocks (Big Bird/ETC, Longformer, Block-wise)
- Customized Kernels (Sparse Transformers, Longformer)

UWNLP AI2

Pre-specified sparsity patterns

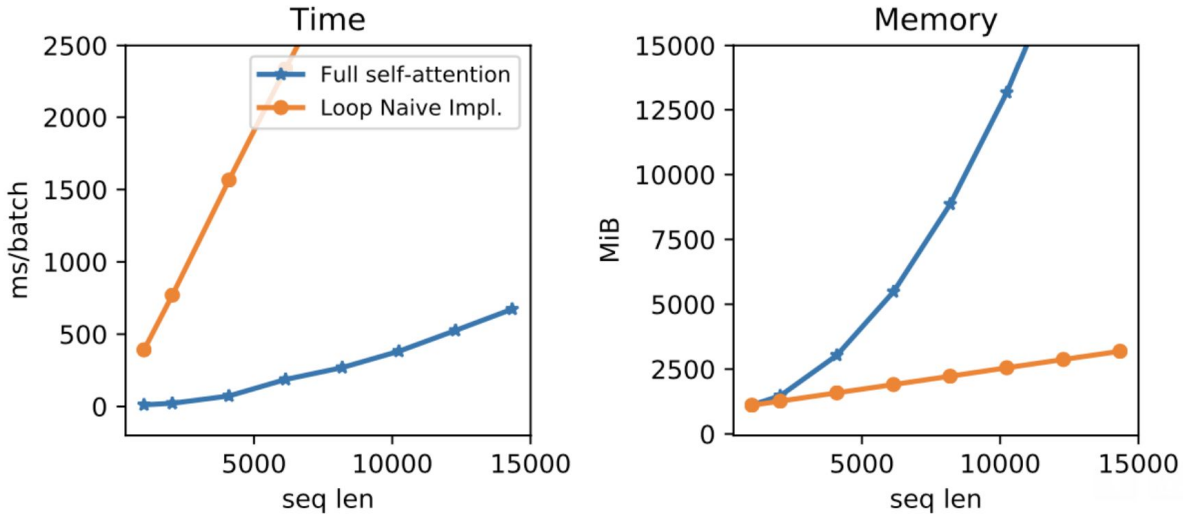
Implementation of sparsity

Longformer (Beltagy, et al., 2020)
Efficient implementation of sparsity using blocks
More computation than needed, but ensures symmetric local attention windows



Sparsifying the attention

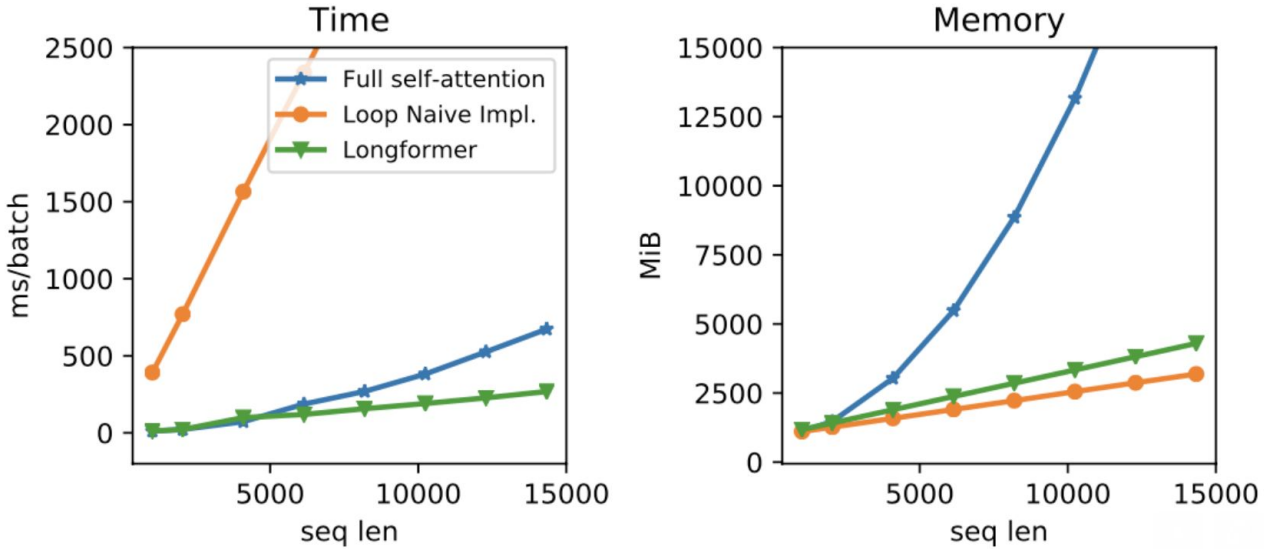
Longformer (Beltagy et al., 2020)



UWNLP AI2

Sparsifying the attention

Longformer (Beltagy et al., 2020)

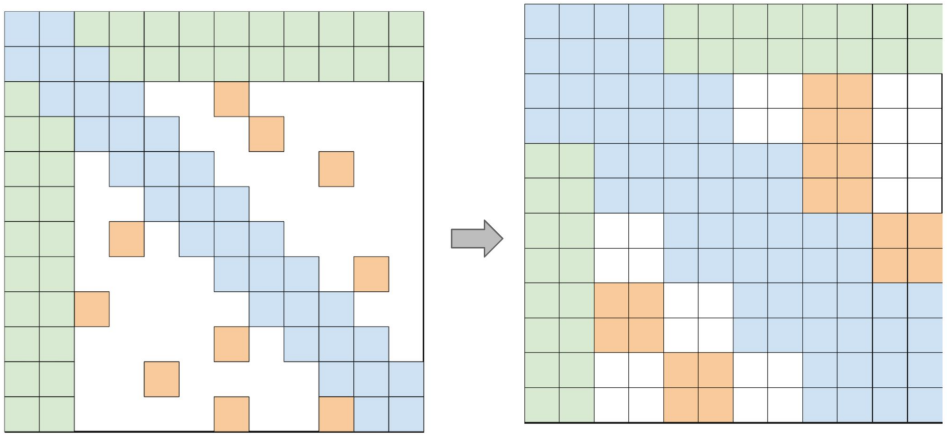


UWNLP AI2

Pre-specified sparsity patterns

Implementation of sparsity

Big Bird (Zaheer et al., 2020)



Efficient blockified implementation

Reduce gather operation

Non-symmetric attention windows

UWNLP AI2

Pre-specified sparsity patterns

Implementation of sparsity - Customized Kernels

- Can support any sparsity pattern
- Difficult to implement
- Hardware specific

Block Sparse (Gray et al., 2017) used by Sparse Transformers

- Only supports specific versions of TF

TVM toolkit (Chen et al., 2018) used by Longformer

- Only supports specific cuda versions

UWNLP AI2

Sparsifying attention - Results

Longformer (Beltagy et al., 2020)

Downstream performance on NLP tasks

	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**

Model	WikiHop	TriviaQA	HotpotQA
Current* SOTA	78.3	73.3	**74.2**
Longformer-large	**81.9**	**77.3**	73.2

UWNLP AI2

Sparsifying attention - Results

Big Bird (Zaheer et al., 2020)

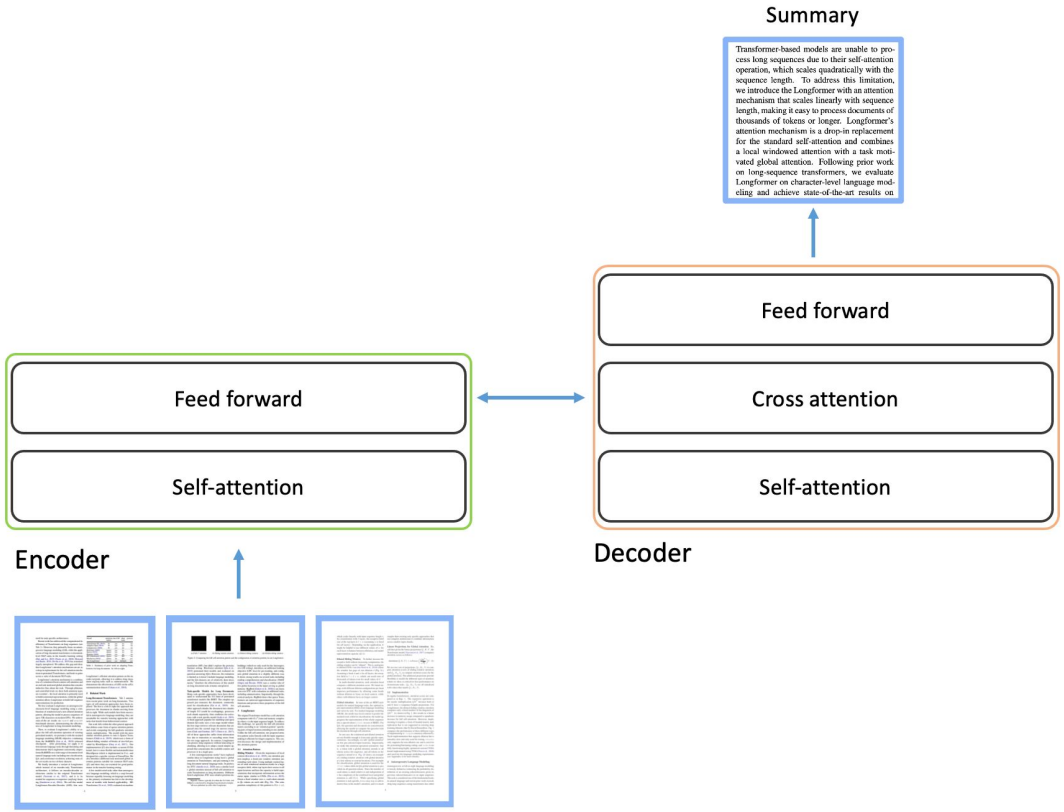
Model	HotpotQA			NaturalQ		TriviaQA		WikiHop
	Ans	Sup	Joint	LA	SA	Full	Verified	MCQ
HGN [26]	**82.2**	88.5	**74.2**	-	-	-	-	-
GSAN	81.6	88.7	73.9	-	-	-	-	-
ReflectionNet [32]	-	-	-	77.1	**64.1**	-	-	-
RikiNet-v2 [61]	-	-	-	76.1	61.3	-	-	-
Fusion-in-Decoder [39]	-	-	-	-	-	84.4	90.3	-
SpanBERT [42]	-	-	-	-	-	79.1	86.6	-
MRC-GCN [87]	-	-	-	-	-	-	-	78.3
MultiHop [14]	-	-	-	-	-	-	-	76.5
Longformer [8]	81.2	88.3	73.2	-	-	77.3	85.3	81.9
BIGBIRD-ETC	81.2	**89.1**	73.6	**77.8**	57.9	**84.5**	**92.4**	**82.3**

UWNLP AI2

Sparsifying attention

Encoder-decoder setting

Important for seq2seq NLP tasks (e.g., summarization, translation)

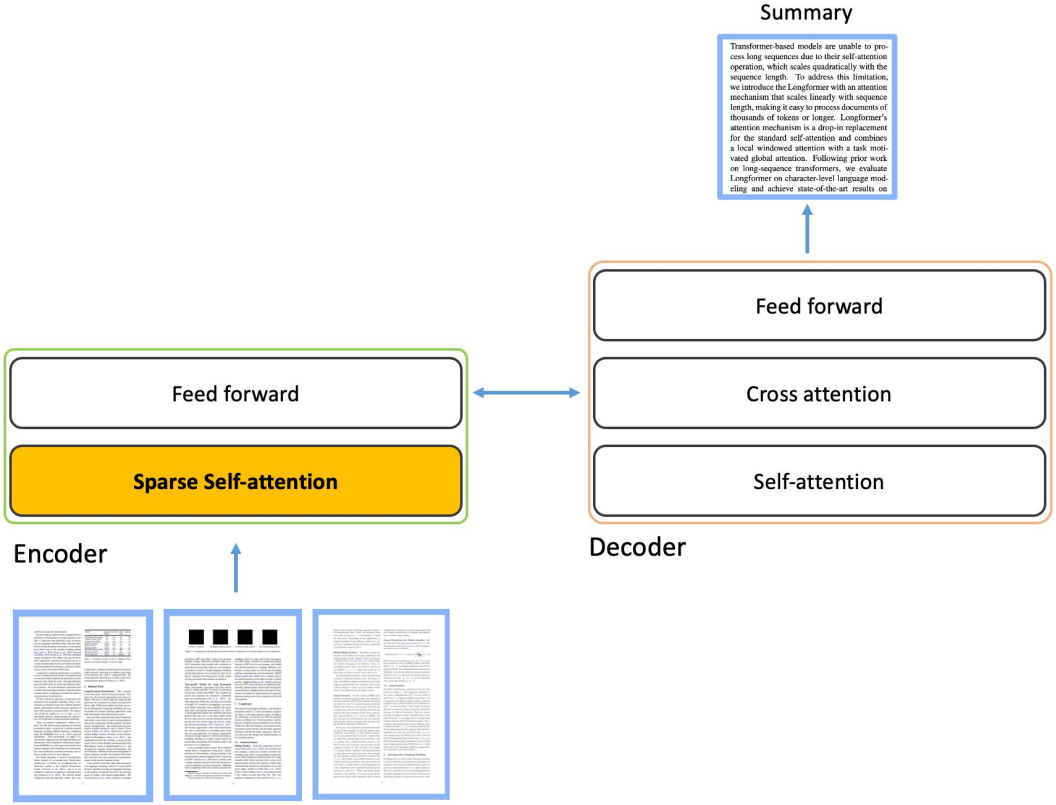


Sparsifying attention

Big Bird and Longformer (LED)

Replace encoder self-attention with sparse attention

Decoder self-attention and cross attention are still full
- Summarization output is short



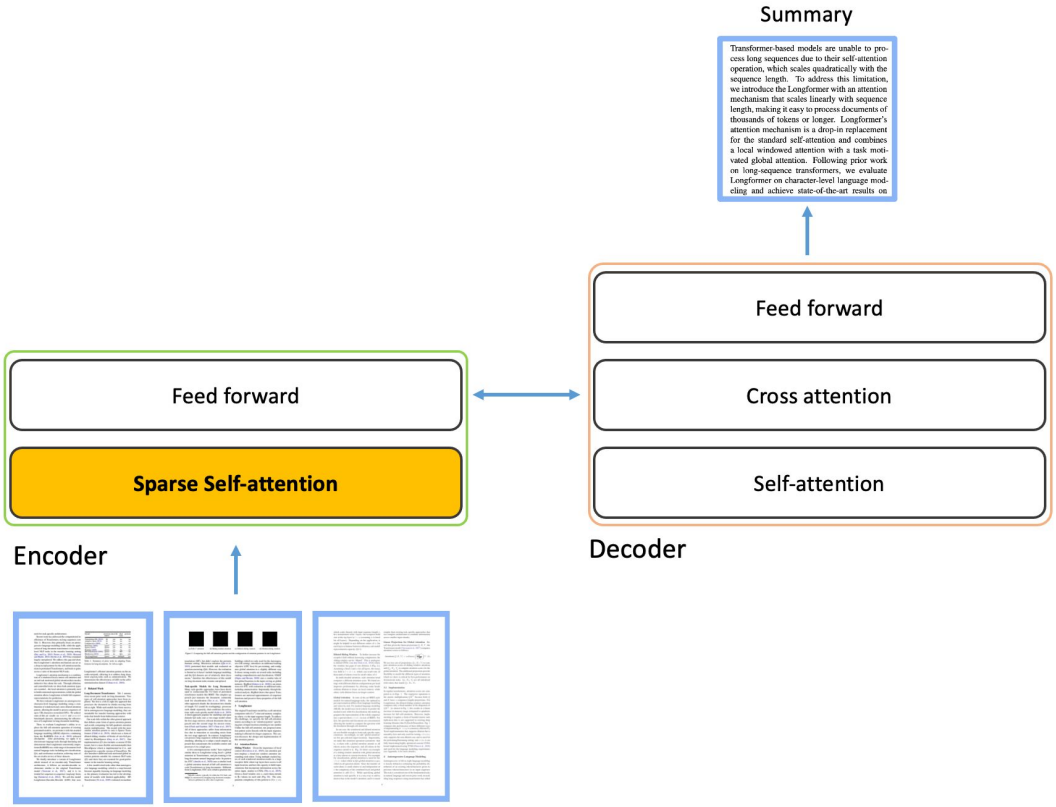
Sparsifying attention

Big Bird

Summarization-specific pretraining using Pegasus tasks (Zhang et al., 2019)

Longformer Encoder Decoder (LED)

Start from BART (Lewis et al., 2019)
Extends position embeddings by copying
No additional pre-training



Sparsifying attention

Big Bird summarization results

Ability to process entire long documents results in SOTA on multiple datasets

	Model	Arxiv			PubMed			BigPatent		
		R-1	R-2	R-L	R-1	R-2	R-L	R-1	R-2	R-L
Prior Art	Attn-Seq2Seq	29.30	6.00	25.56	31.55	8.52	27.38	28.74	7.87	24.66
	Pntr-Gen-Seq2Seq	32.06	9.04	25.16	35.86	10.22	29.69	33.14	11.63	28.55
	Long-Doc-Seq2Seq	35.80	11.05	31.80	38.93	15.37	35.21	-	-	-
	Sent-CLF	34.01	8.71	30.41	45.01	19.91	41.16	36.20	10.99	31.83
	Sent-PTR	42.32	15.63	38.06	43.30	17.92	39.47	34.21	10.78	30.07
	Extr-Abst-TLM	41.62	14.69	38.03	42.13	16.27	39.21	38.65	12.31	34.09
	Dancer	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	31.98	8.13	29.53	35.77	13.85	33.32	41.11	22.10	32.58
	+ Pegasus	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)	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**

UWNLP AI2

Sparsifying attention

Longformer (LED)
summarization results

Strong results by processing 16K token long documents without further pretraining

	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)	44.40	17.94	39.76
BigBird (seqlen: 4,096) (2020)	**46.63**	19.02	41.77
LED-large (seqlen: 16,384)	**46.63**	**19.62**	**41.83**

UWNLP AI2

Approximating attention and kernel methods

Kernels and approximating self-attention

Intuition from **Linear Transformer** (Katharopoulos et al. 2020)

Recall self-attention formula $$A(X) = V' = \text{softmax}(\frac{QK^T}{\sqrt{d}})V$$

Could rewrite as following: $$V'_i = \frac{\sum_{j=1}^{n} \text{sim}(Q_i, K_j)V_j}{\sum_{j=1}^{n} sim(Q_i, K_j)}$$

Instead of dot product for similarity, use a kernel function: $\text{sim}(a, b) = \phi(a).\phi(b)$

$$V'_i = \frac{\sum_{j=1}^{n} \phi(Q_i)^T \phi(K_j)V_j}{\sum_{j=1}^{n} \phi(Q_i).\phi(K_j)}$$

UWNLP AI2

Kernels and approximating self-attention

Instead of dot product for similarity, use a kernel function:

$$V_i' = \frac{\sum_{j=1}^{n} \phi(Q_i)^T \phi(K_j) V_j^T}{\sum_{j=1}^{n} \phi(Q_i).\phi(K_j)} \Rightarrow V_i' = \frac{\phi(Q_i)^T \sum_{j=1}^{n} \phi(K_j) V_j^T}{\phi(Q_i) \sum_{j=1}^{n} \phi(K_j)}$$

Associative property of matrix multiplication

The cost becomes linear as $\sum_{j=1}^{N} \phi(K_j) V_j^T$ can be computed once and reused

Feature map based on the exponential linear unit activation (elu)

UWNLP AI2

Kernels and approximating self-attention

Linear Transformer (Katharopoulos et al. 2020)

How can this be extended to autoregressive transformers?

Causal masking

Non-autoregressive

$$V_i' = \frac{\sum_{j=1}^{N} \text{sim}(Q_i, K_j)\, V_j}{\sum_{j=1}^{N} \text{sim}(Q_i, K_j)}$$

Autoregressive

$$V_i' = \frac{\sum_{j=1}^{i} \text{sim}(Q_i, K_j)\, V_j}{\sum_{j=1}^{i} \text{sim}(Q_i, K_j)}$$

* Some of the linear Transformer slides are from original author slides

UWNLP AI2

Kernels and approximating self-attention

Linear Transformer (Katharopoulos et al. 2020)

However, both S_i and Z_i can be computed as cumulative sums (constant time) given the previous values

Allows recovering linear complexity

Non-autoregressive

$$V'_i = \frac{\phi(Q_i)^T \overbrace{\sum_{j=1}^{N} \phi(K_j) V_j^T}^{S}}{\phi(Q_i)^T \underbrace{\sum_{j=1}^{N} \phi(K_j)}_{Z}}$$

Autoregressive

$$V'_i = \frac{\phi(Q_i)^T \overbrace{\sum_{j=1}^{i} \phi(K_j) V_j^T}^{S_i}}{\phi(Q_i)^T \underbrace{\sum_{j=1}^{i} \phi(K_j)}_{Z_i}}$$

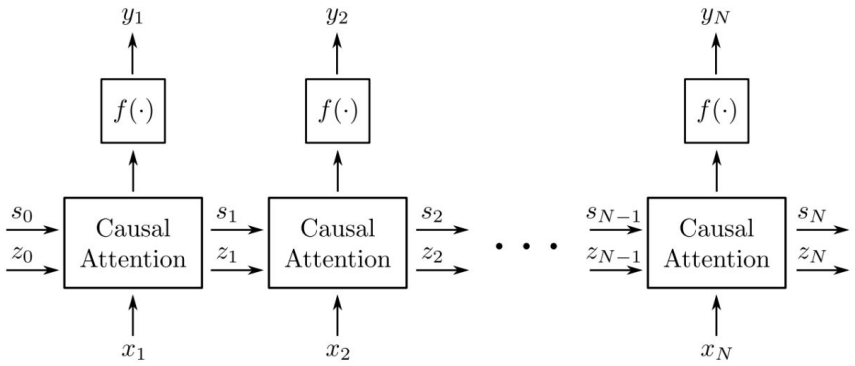
Naive computation of S_i and Z_i results in quadratic complexity.

UWNLP AI2

Kernels and approximating self-attention

Linear Transformer (Katharopoulos et al. 2020)

Autoregressive transformers can be written as a function that **receives an input** x_i, **modifies the internal state** $\{s_{i-1}, z_{i-1}\}$ and **predicts an output** y_i.

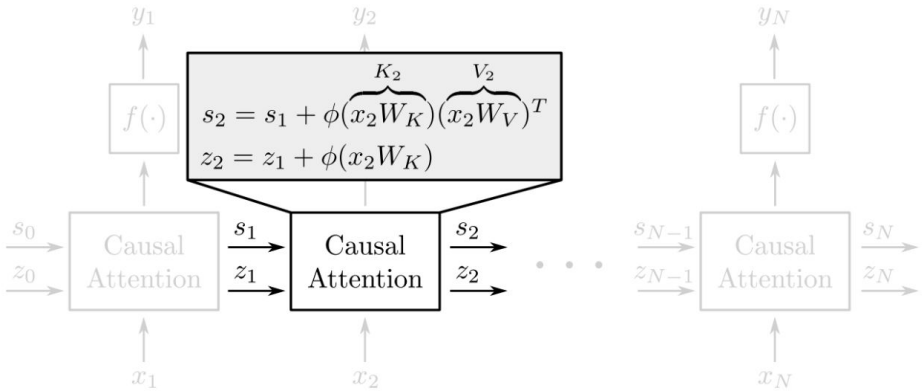


Autoregressive inference with **linear complexity and constant memory**.

Kernels and approximating self-attention

Linear Transformer (Katharopoulos et al. 2020)

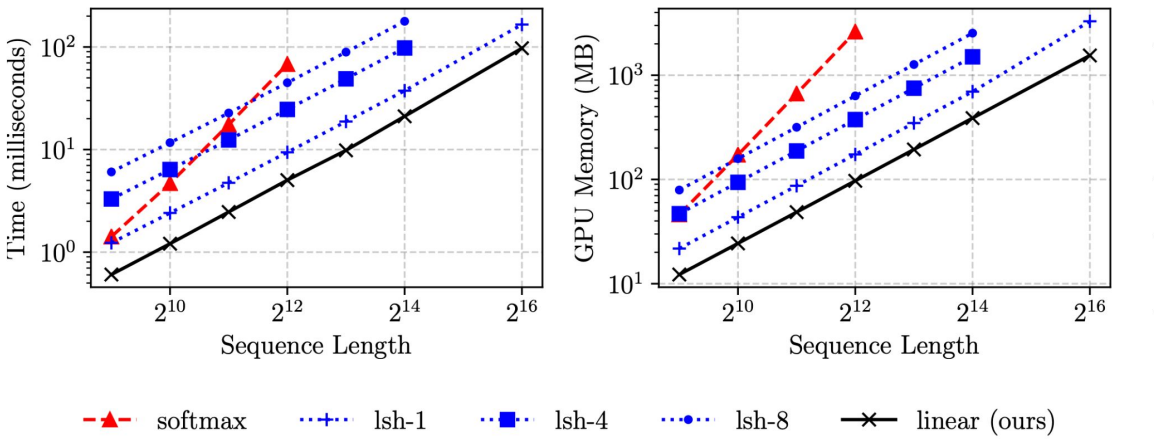
Autoregressive transformers can be written as a function that **receives an input** x_i, **modifies the internal state** $\{s_{i-1}, z_{i-1}\}$ and **predicts an output** y_i.



UWNLP AI2

Kernels and approximating self-attention

Linear Transformer (Katharopoulos et al. 2020)

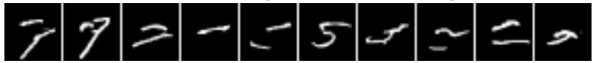


Unconditional samples after 250 epochs on MNIST

Ours (0.644 bpd)

Softmax (0.621 bpd)

LSH-1 (0.745 bpd)

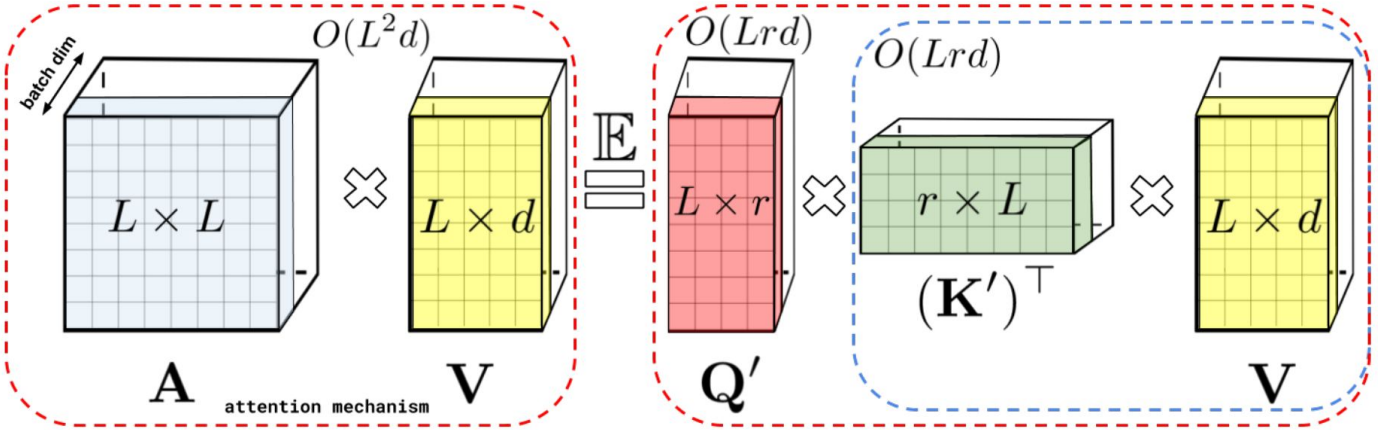


LSH-4 (0.676 bpd)

Kernels and approximating self-attention

Performer (Choromanski et al., 2020)

Generalized attention mechanizm and random kernels

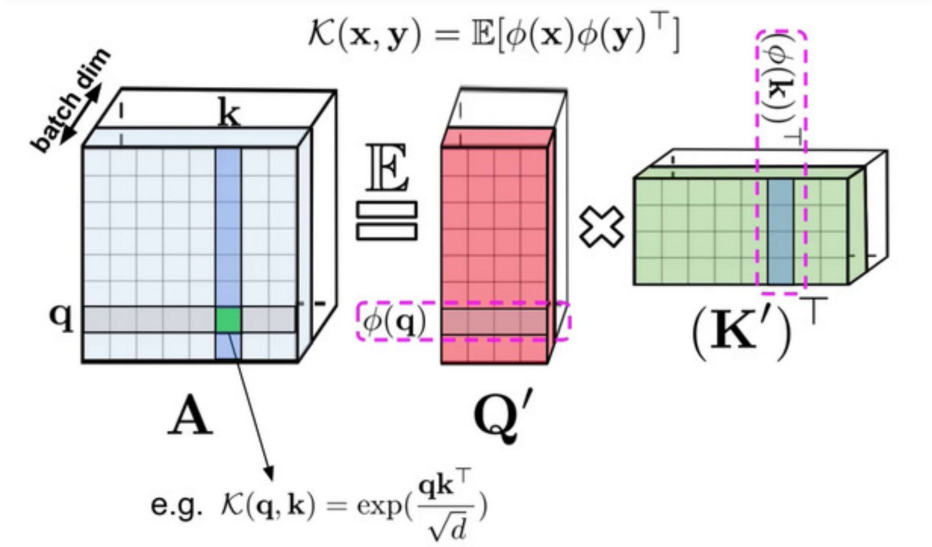


* Some of Performer related slides are taken from the original slides by the authors

Estimating full self-attention

Performer (Choromanski et al., 2020)

Attention is Kernelizable

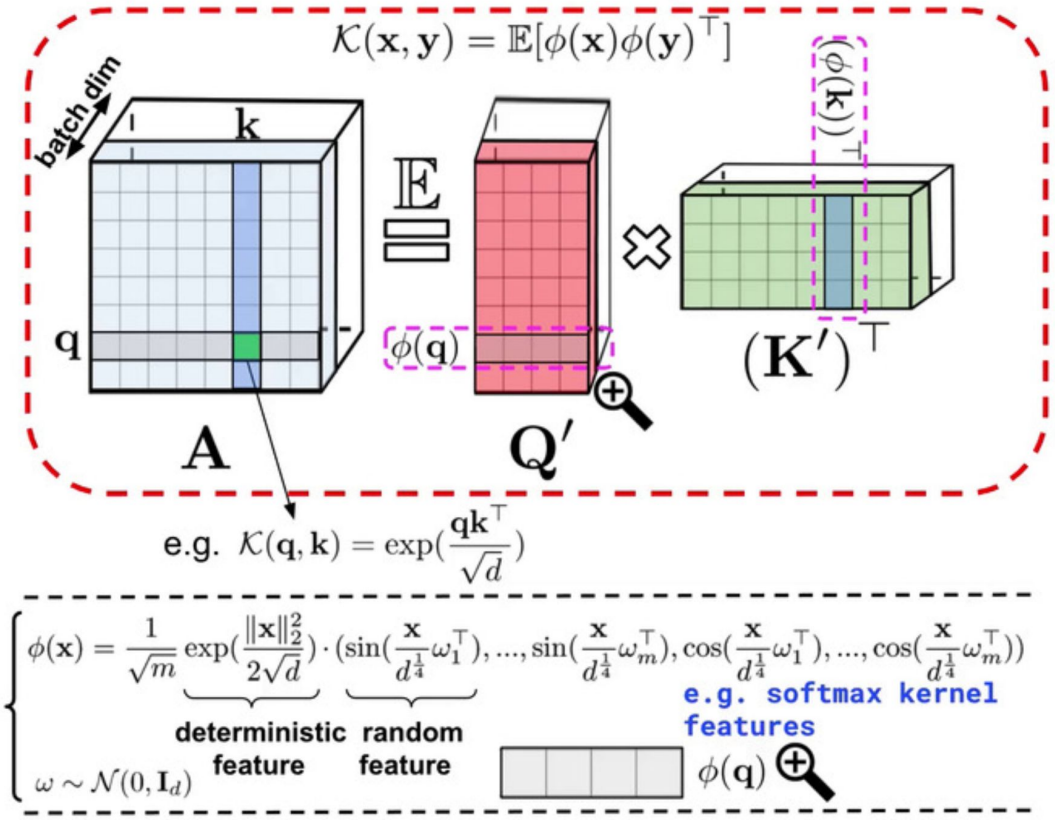


Kernels and approximating self-attention

Performer (Choromanski et al., 2020)

Random Fourier features

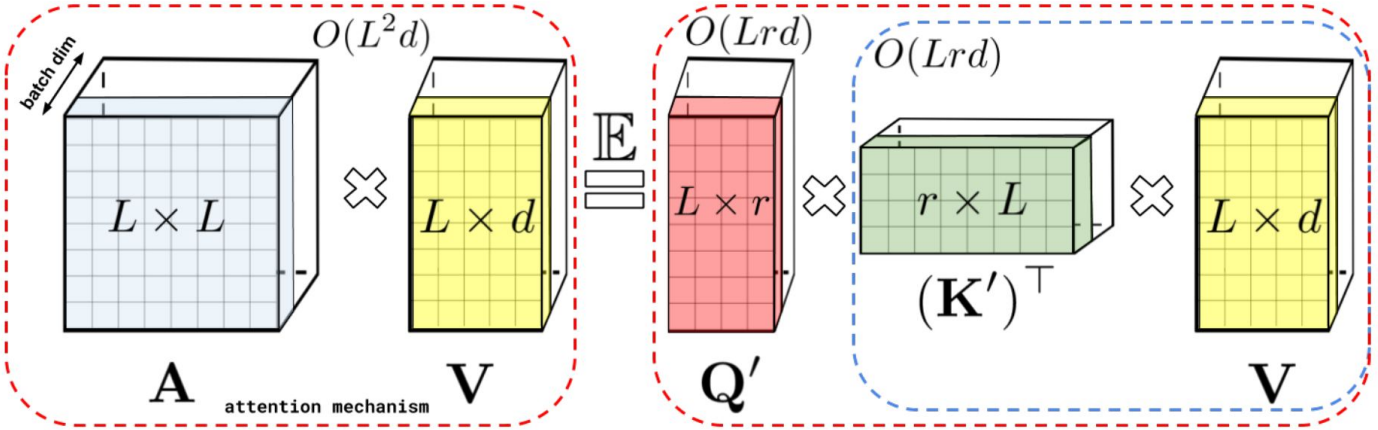
FAVOR+ algorithm



Kernels and approximating self-attention

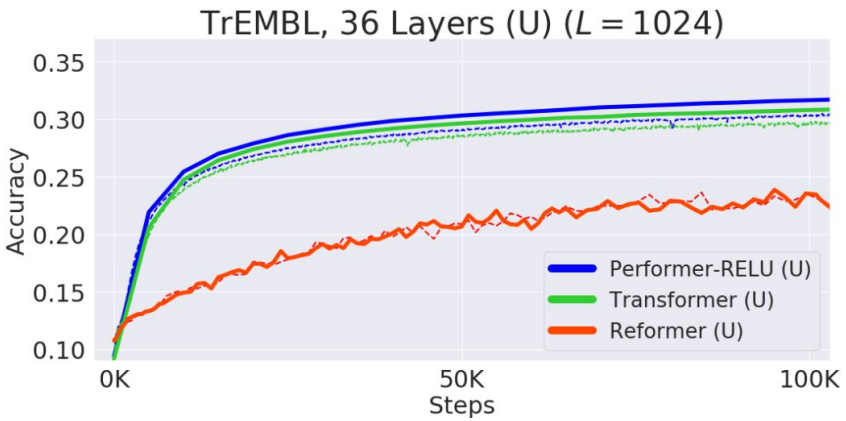
Performer (Choromanski et al., 2020)

In practice the number of features is much lower than the sequence length

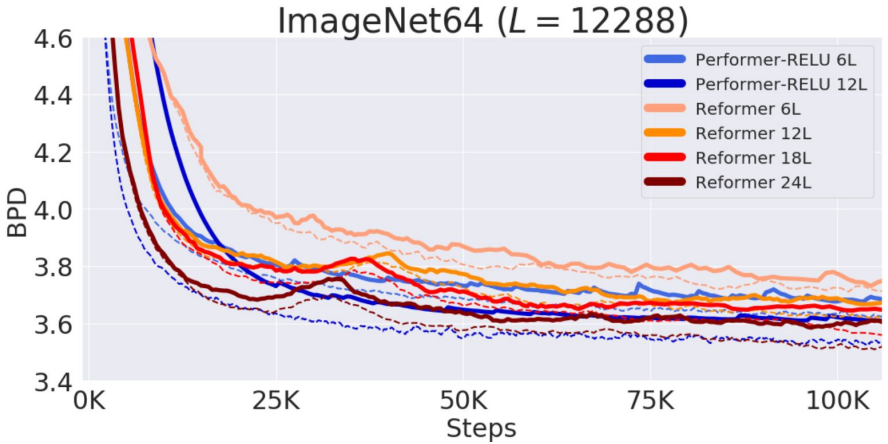


Kernels and approximating self-attention

Performer (Choromanski et al., 2020)



Modeling protein sequence



Generating images

UWNLP AI2

Kernels and approximating self-attention

Random Feature Attention (Peng et al., 2021)

- Kernelizing attention similar to Linear Transformer
- Instead of exponential linear unit feature map, uses random features

$$\boldsymbol{\phi}(\mathbf{x}) = \sqrt{1/D}\left[\sin(\mathbf{w}_1 \cdot \mathbf{x}), \ldots, \sin(\mathbf{w}_D \cdot \mathbf{x}), \cos(\mathbf{w}_1 \cdot \mathbf{x}), \ldots, \cos(\mathbf{w}_D \cdot \mathbf{x})\right]^\top.$$

- D dimensional random vectors w_i are sampled from

$$\mathbb{E}_{\mathbf{w}_i}\left[\boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{y})\right] = \exp\left(-\|\mathbf{x}-\mathbf{y}\|^2 / 2\sigma^2\right)$$

UWNLP AI2

Kernels and approximating self-attention

Random Feature Attention (Peng et al., 2021)

- Concurrent work with Performer
- Performer uses positive random features to approximate softmax, aiming for a lower variance in critical regions.
- RFA instead normalizes the queries and keys before random projection to reduce variance

Kernels and approximating self-attention

Random Feature Attention (Peng et al., 2021)

	Small		Big	
Model	**Dev.**	**Test**	**Dev.**	**Test**
BASE	33.0	34.5	24.5	26.2
ϕ_{elu} (Katharopoulos et al., 2020)	38.4	40.1	28.7	30.2
RFA-Gaussian	33.6	35.7	25.8	27.5
RFA-arccos	36.0	37.7	26.4	28.1
RFA-GATE-Gaussian	**31.3**	**32.7**	**23.2**	**25.0**
RFA-GATE-arccos	**32.8**	**34.0**	24.8	26.3
RFA-GATE-Gaussian-Stateful	**29.4**	**30.5**	**22.0**	**23.5**

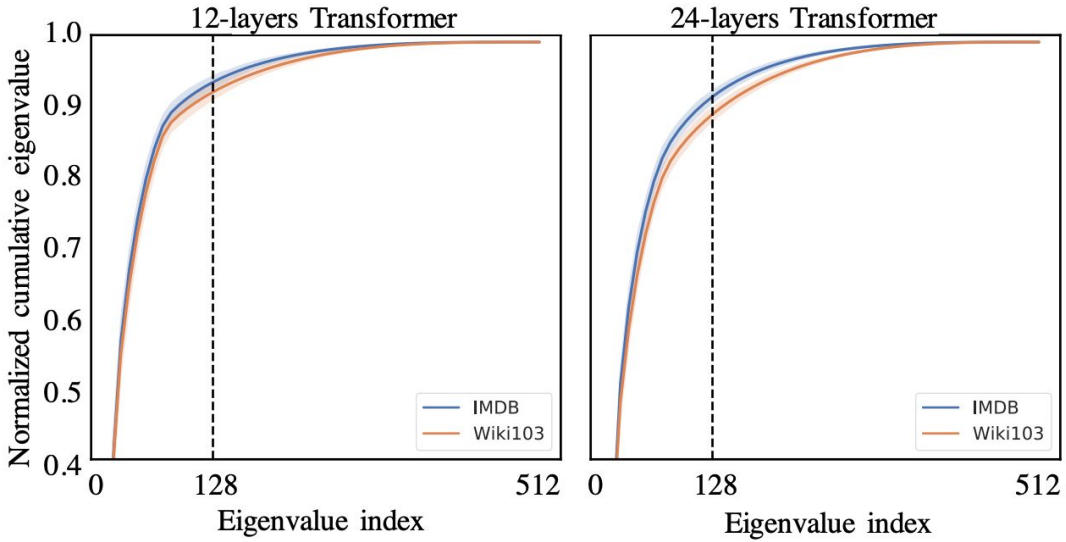
LM Preplexity on Wikitext 103

UWNLP AI2

Kernels and approximating self-attention

Linformer (Wang et al 2020)

Self-attention matrix is low-rank



Apply SVD on matrix P (self-attention)

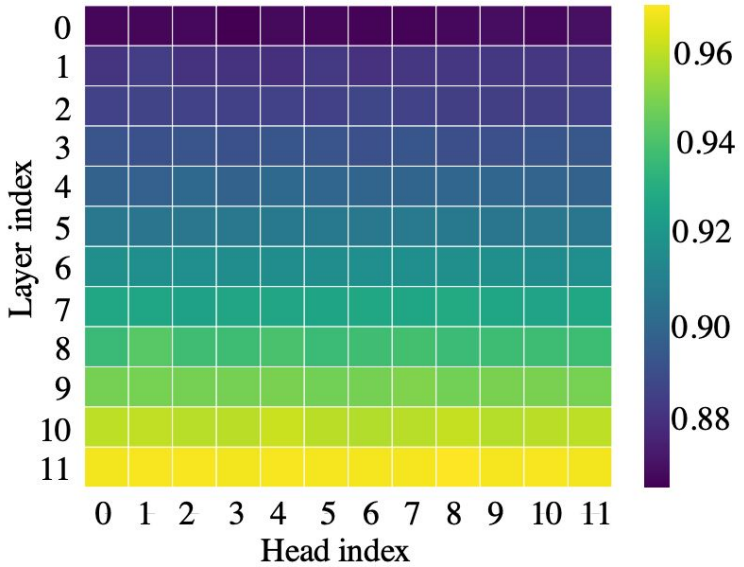
Most of the information of matrix P can be recovered from the first few largest singular values

UWNLP AI2

Kernels and approximating self-attention

Linformer (Wang et al 2020)

Rank in higher layers is even lower



heatmap of the normalized cumulative singular value at the 128-th largest singular value

UWNLP AI2

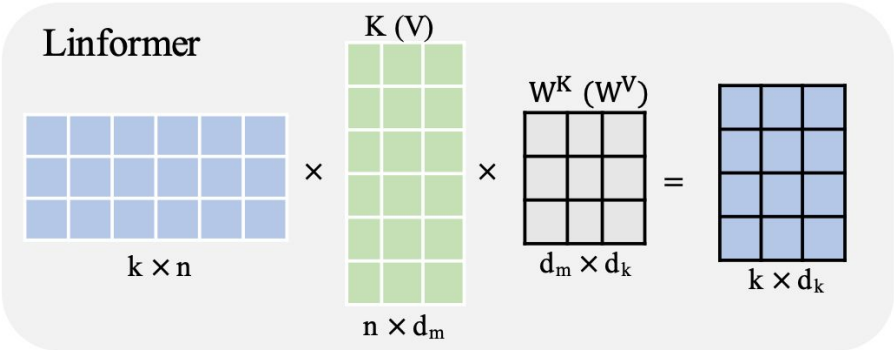
Kernels and approximating self-attention

Linformer (Wang et al 2020)

Low-rank approximations of the self-attention matrix

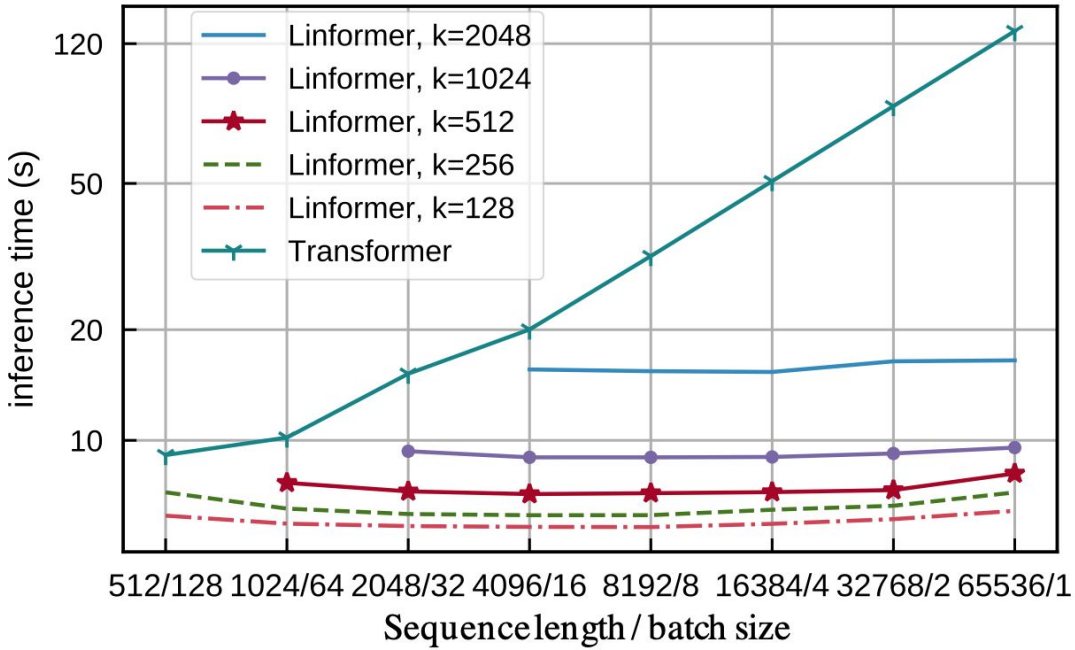
Projects the length dimension of keys and values to a lower-dimensional representation (N → k) [N: seq len]

Reduce O(N^2) to O(Nk)



Kernels and approximating self-attention

Linformer (Wang et al 2020)



Kernels and approximating self-attention

Linformer (Wang et al 2020)

When pre-trained, competitive performance with full self-attention

n	Model	SST-2	IMDB	QNLI	QQP	Average
	Liu et al. (2019), RoBERTa-base	93.1	94.1	90.9	**90.9**	92.25
	Linformer, 256, shared kv, layer	93.1	94.1	**91.2**	90.8	**92.30**
512	Devlin et al. (2019), BERT-base	92.7	93.5	91.8	89.6	91.90
	Sanh et al. (2019), Distilled BERT	91.3	92.8	89.2	88.5	90.45
1024	Linformer, 256	93.0	93.8	90.4	90.4	91.90
	Linformer, 256, shared kv	93.0	93.6	90.3	90.4	91.83
	Linformer, 256, shared kv, layer	93.2	**94.2**	90.8	90.5	92.18

UWNLP AI2

Benchmarking Transformers

Benchmarking Transformers

Long Range Arena (Tay et al., 2020)

A suite of 6 tasks to evaluate transformers, without the need for pre-training

1- Long list ops **INPUT:** [MAX 4 3 [MIN 2 3] 1 0 [MEDIAN 1 5 8 9, 2]] **OUTPUT:** 5

2- Byte-level text classification

3- Byte-level document retrieval

4- Image classification on sequence of pixels

UWNLP AI2

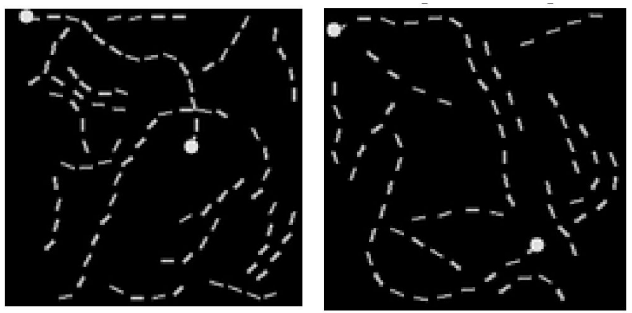
Benchmarking Transformers

Long Range Arena (Tay et al., 2020)

A suite of 6 tasks to evaluate transformers, without the need for pre-training

5- Pathfinder: binary classification whether two points represented are connected by a path consisting of dashes
32x32 images (1024 max seq len)

6- Pathfinder-X
128x128 images (16K seq len)

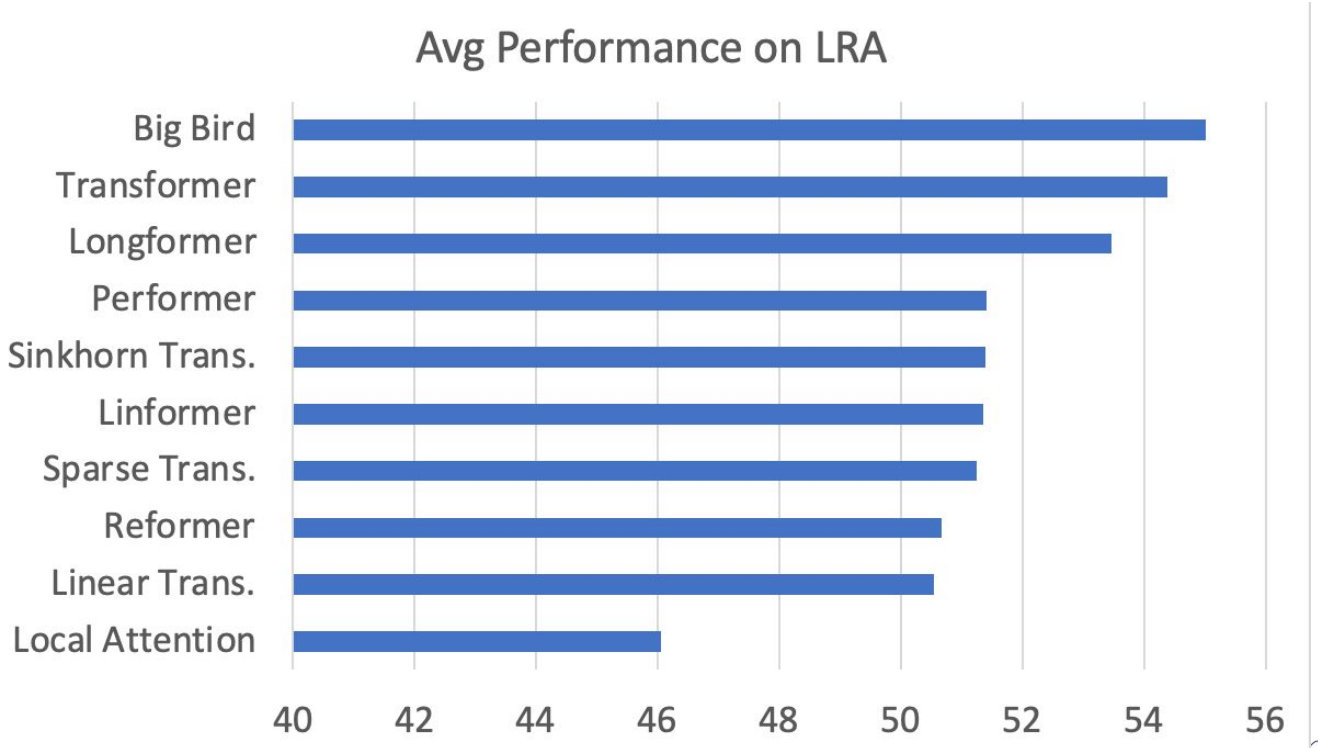


(a) A positive example.

(b) A negative example.

UWNLP AI2

Benchmarking Transformers



Benchmarking Transformers

Note: these results might be sensitive to implementation details, hardware and hyper-parameters.

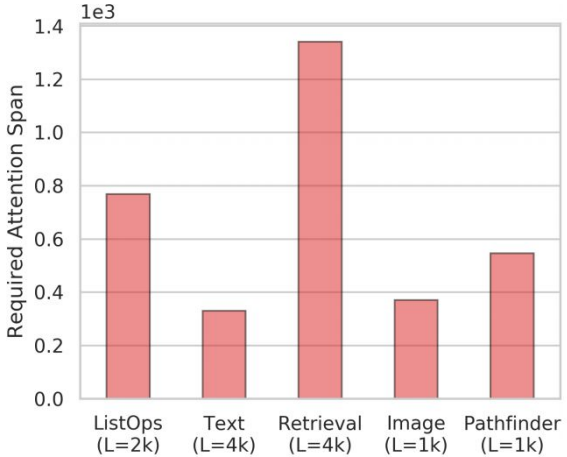
Benchmarking Transformers

Long Range Arena (Tay et al., 2020)

A nice benchmark for probing and analyses of Transformer models

Eliminates the need for expensive pre-training

Most tasks are synthetic, might not reflect behavior on real downstream tasks



Required attention span on different tasks.

Key takeaways

- Many approaches for making transformers efficient for long sequences
  - Sparse Patterns
  - Kernels and approximations
- Most of these methods can provide attention computation in linear time without notable performance loss, allowing scaling to long sequences
- Many of these approaches are complementary to each other and can be combined for more efficient models
- While many works have been introduced, few of them conduct extensive analyses on downstream NLP tasks (next section provides more details).

UWNLP AI2