Efficient Transformers

김백수 이다인 조주현

TABLE OF CONTENTS

01 Efficient Transformer

02 Pattern Methods

03 Memory & Recurrence

04 Low Rank & Kernels

Efficient Transformer

Efficient Transformer

Efficient Transformers: A Survey

Yi Tay YITAY@GOOGLE.COM
Google Research

Mostafa Dehghani DEHGHANI@GOOGLE.COM
Google Research, Brain team

Dara Bahri DBAHRI@GOOGLE.COM
Google Research

Donald Metzler METZLER@GOOGLE.COM
Google Research

Editor: Preprint

Efficient Transformer

Motivation

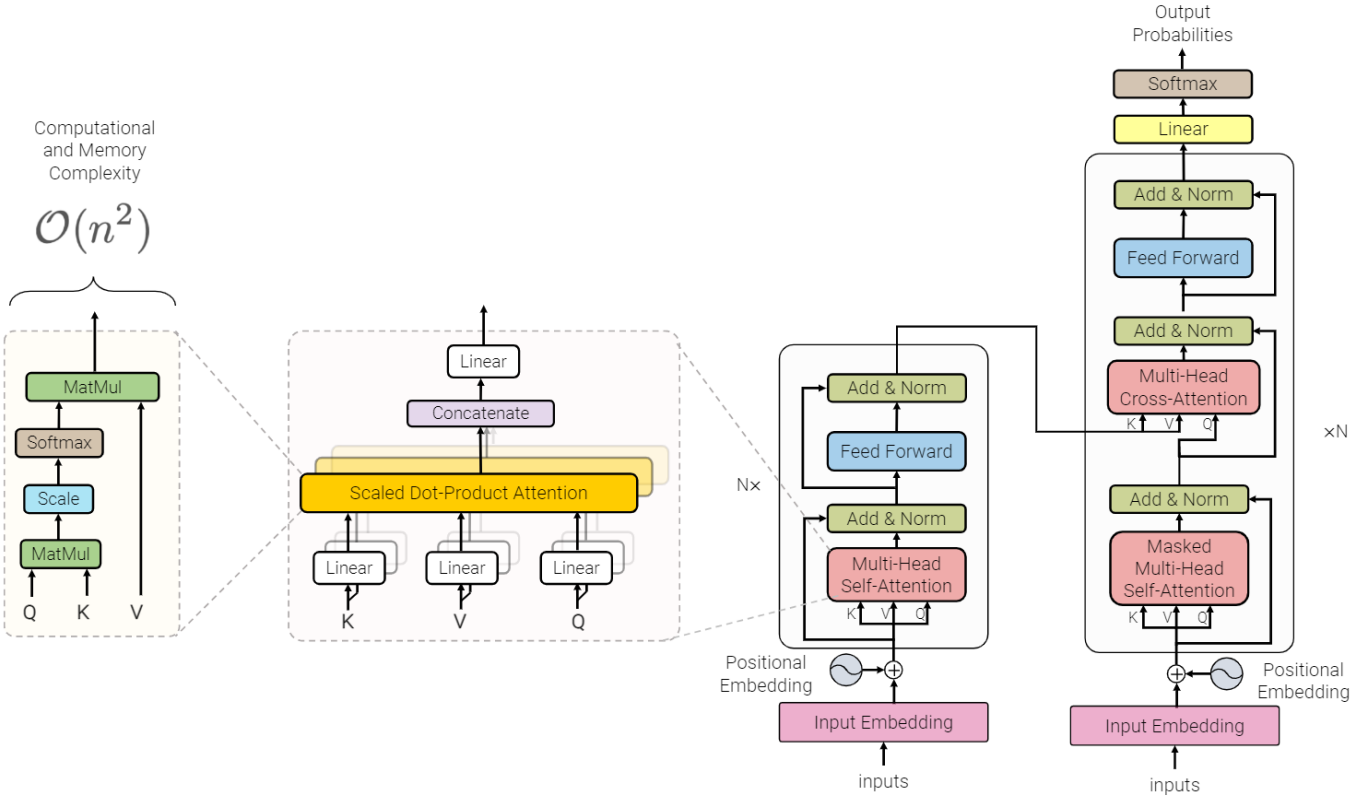
- Attention을 계산하기 위한 Matrix Multiplication 에서 오는 quadratic complexity
- Transformer 의 Scalability 를 늘리기 위한 시도들
- Longer Sequence, Less Memory

Transformer

*Transformer blocks are characterized by a **multi-head self-attention** mechanism, a **positionwise feed-forward** network, **layer normalization** (Ba et al., 2016) modules and **residual connectors**.*

- Multi-head (self) attention
- Positionwise FFN
- Layer Normalization
- Residual connectors

Transformer



Multi-Head Attention

*The key idea behind the mechanism is to learn an **alignment** (Bahdanau et al., 2014) in which each element in the sequence **learns to gather from other tokens in the sequence***

다른 토큰 간의 관계로서 피쳐를 임베딩

Scaled Dot-Product Multi-Head Attention(Calculation)

The key idea behind the mechanism is to learn an ***alignment*** *(Bahdanau et al., 2014) in which each element in the sequence* ***learns to gather from other tokens in the sequence***

$$A_h = Softmax(\alpha Q_h {K_h}^T) V_h$$

$Q, K, V : LinearTransform(X) : R^N \times R^{d/N_h}$

$X(input): R^N \times R^d$

$N: Sequence\ Length$, $d: Feature\ Dimension$, $N_h: Nubmer\ of\ Attention\ Heads$

α: Scaling Factor

On the scalability of Self-Attention

$$A_h = Softmax(\alpha Q_h {K_h}^T) V_h$$

Q, K, V : $LinearTransform(X) : R^N \times R^{d/N_h}$

$Assume\ N_h = 1$

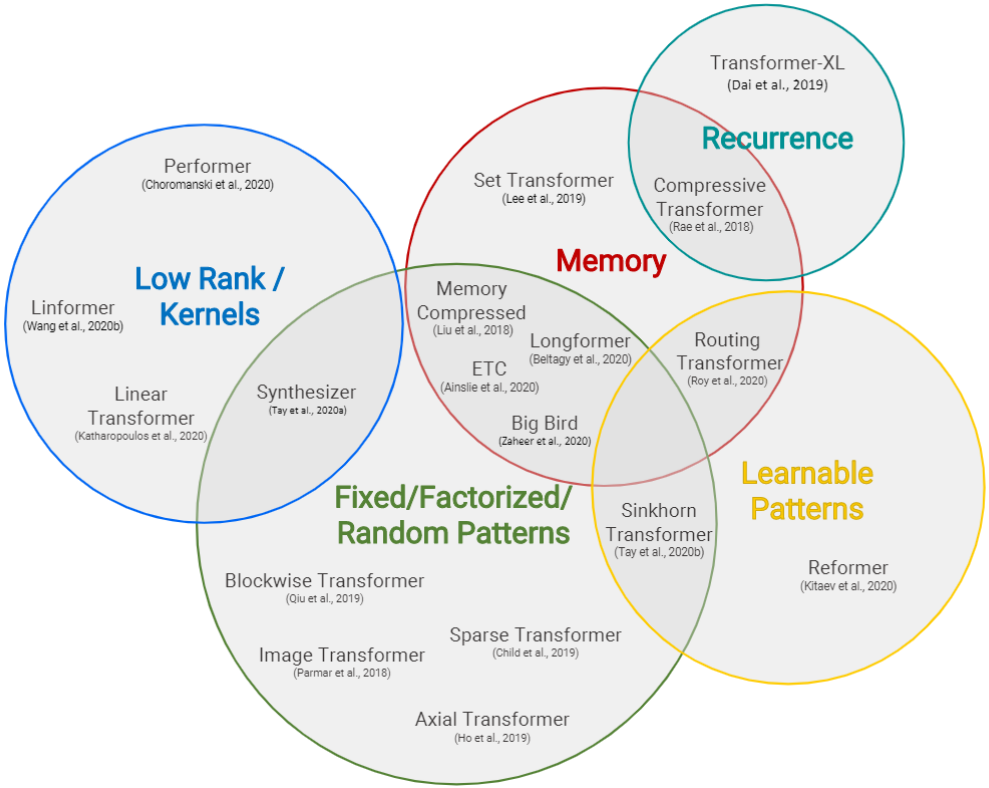
${Q_h}{K_h}^T \sim O(N^2 d)$

$Softmax(\sim)V_h \sim O(N^2 d)$

N 이 매우 길 때

→ Quadratic to N!

Efficient Transformers - Taxonomy



Efficient Transformers

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

Abstract

Patterns: Sequence 중 일부 Pattern(Block)만 Attention을 보자.

Memory: 전체 Sequence 와 Global Attention 을 보는 Memory를 따로 두자.

Low Rank: Attention 연산에 사용되는 Matrix를 낮은 Rank로 압축하자.

Kernels: 커널 근사를 통해 NxN Matrix Calculation을 생략하여 계산 복잡도를 줄이자.

Recurrence: 시퀀스의 앞부분은 낮은 Rank의 State로 재귀압축하여 사용하자.

Pattern Methods

Pattern Methods

Patterns: Sequence 중 일부 Pattern만 Attention을 보자.

어떤 패턴?

Fixed Patterns – 고정된 패턴

Combination of Patterns – 고정된 패턴의 조합

Learnable Patterns – 학습가능한 패턴

Fixed Patterns

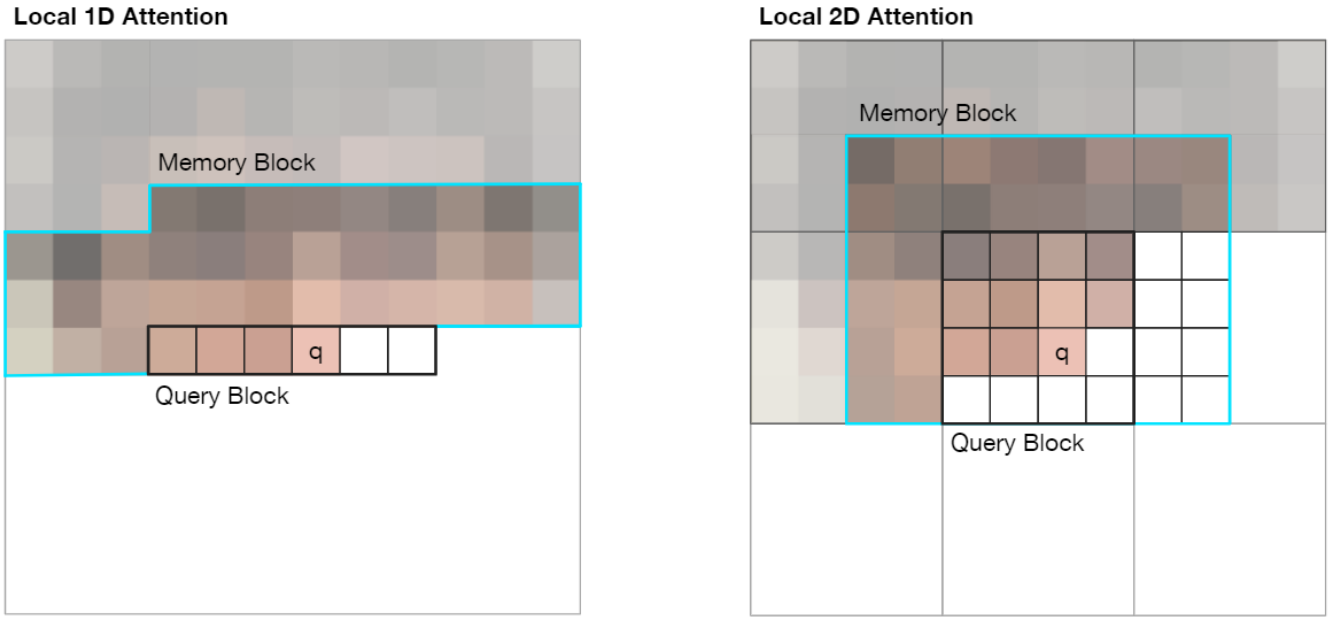
The earlies modifications to self-attention simply sparsifies the attention matrix ***by limiting the field of view*** *to fixed, predefined patterns such as local windows and block patterns of fixed strides*

- Blockwise Patterns
- Strided Patterns
- Compressed Patterns

Image Transformer(2018)

Inspired by convolutional neural networks we address this by adopting a notion of locality, ***restricting the positions in the memory matrix M to a local neighborhood around the query position****.*

Image Transformer(2018)



Designated Memory(Key, Value) Block for Query Blocks

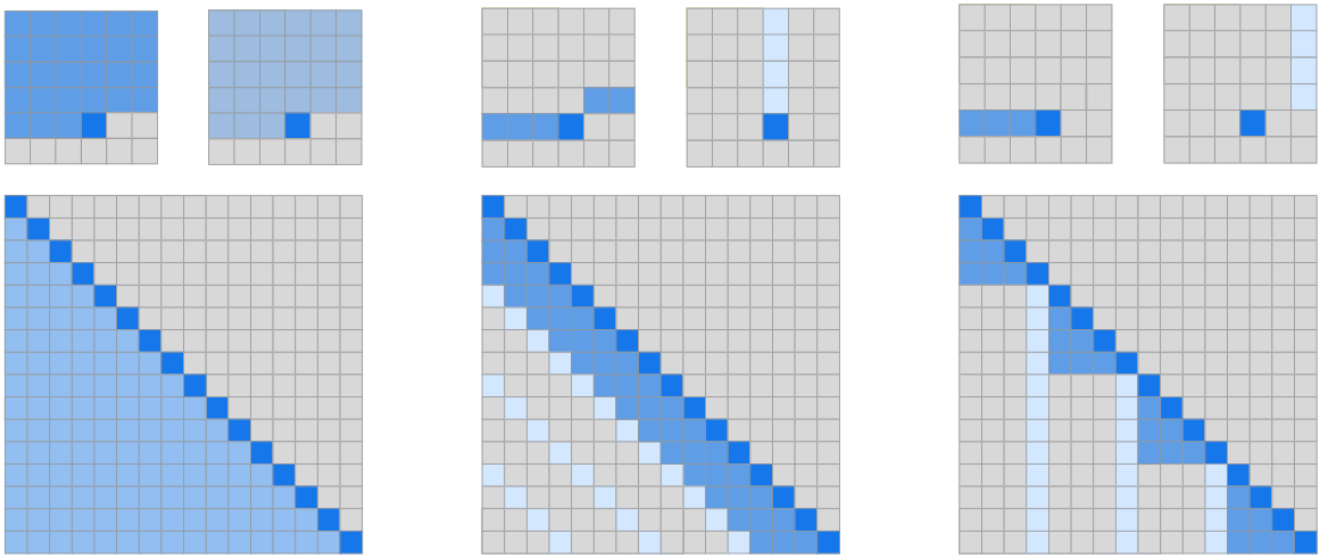
Image Transformer(2018)

$$for\ q1, q2, \ldots (where\ \mathrm{len(q1)} + \mathrm{len(q2)} \ldots = \mathrm{N})$$
$$A_{q_i} = Softmax\left(\alpha Q_{q_i} {K_{q_i}}^T\right) V_{q_i}$$

$Q_{q_i}: Q\left(q_{q_i}\right):\ R^{query-block\ size} \times R^d$
$K_{q_i}: K\left(m_{q_i}\right):\ R^{memory-block\ size} \times R^d$
$V_{q_i}: K\left(m_{q_i}\right):\ R^{memory-block\ size} \times R^d$

$\sim O(NM)\ (where\ M = Memory\ block\ size)$

Sparse Transformer(2019)



(a) Transformer (b) Sparse Transformer (strided) (c) Sparse Transformer (fixed)

Factorized Attention(Multi-Head)

Sparse Transformer(2019)

2 factorization
Local Attention

$$\hat{A}_{ij}\begin{cases} Q_i(K)_j^T, if\ \lfloor j/N \rfloor = \lfloor i/N \rfloor \\ 0 \qquad\qquad\qquad otherwise \end{cases}$$

Strided Attention

$$\hat{A}_{ij}\begin{cases} Q_i(K)_j^T, if(i-j)\ \ mod\ N = 0 \\ 0 \qquad\qquad\qquad\qquad otherwise \end{cases}$$

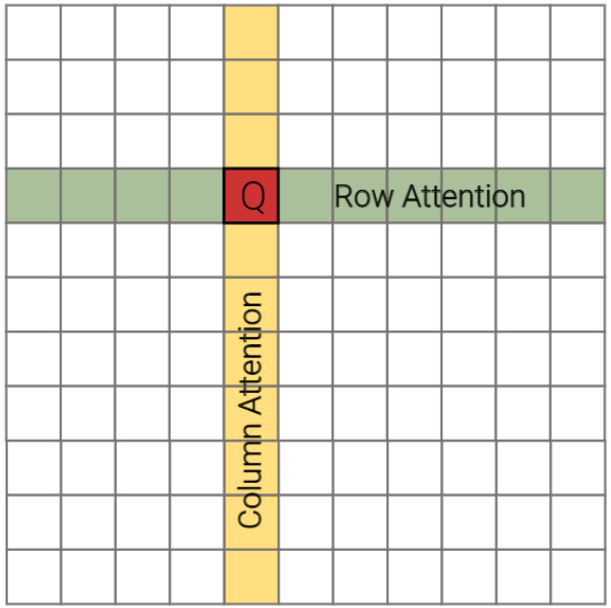
$\sim O\left(N\sqrt{N}\right)$

Sparse Transformer(2019)

This formulation is convenient ***if the data naturally has a structure that aligns with the stride, like images or some types of music****. For data without a periodic structure, like text, however, we find that the network can fail to properly route information with the strided pattern, as spatial coordinates for an element do not necessarily correlate with the positions where the element may be most relevant in the future.*

In those cases, we instead use a ***fixed attention pattern*** *(Figure (c)), where specific cells summarize previous locations and propagate that information to all future cells.*

Axial Transformer(2019)



Attention span in Axial Transformer on a two-dimensional input.

Memory

Memory

- Leverage a side memory module that can access multiple tokens at once.
- A common form is **global memory** which is able to access the entire sequence.
- The **global tokens** act as a form of memory that learns to gather from input sequence tokens.

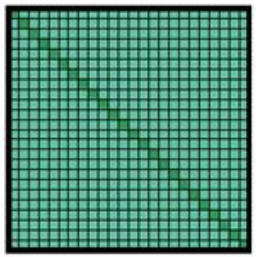
- With a limited number of memory (or inducing points), we are able to perform a preliminary pooling like operation of the input sequence to compress the input sequence

- This was first introduced **in Set Transformers (Lee et al., 2019)** as the inducing points method. These parameters are often interpreted as “memory” and are used as a form of temporary context for future processing.

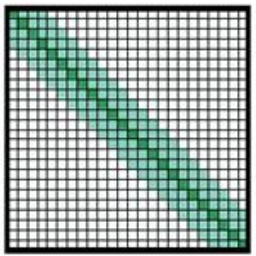
- This can be thought of as a form of parameter attention (Sukhbaatar et al., 2019). Global memory is also used in ETC (Ainslie et al., 2020) and Longformer (Beltagy et al., 2020).

Longformer(2020)

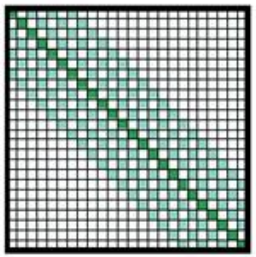
- Variant of **Sparse Transformer**
- Improves Sparse Transformer with "**Dilated Sliding Windows**"
  ✓which can enable better long-range coverage without sacrificing sparsity.
- For classification tasks, the Longformer adopts **global tokens** (e.g., CLS tokens) that have access to all input sequences.
- The complexity of the model is reduced from O(n^2) to O(nk)
  ✓where k is the size of the window



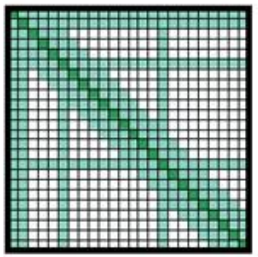
(a) Full n^2 attention



(b) Sliding window attention



(c) Dilated sliding window



(d) Global+sliding window

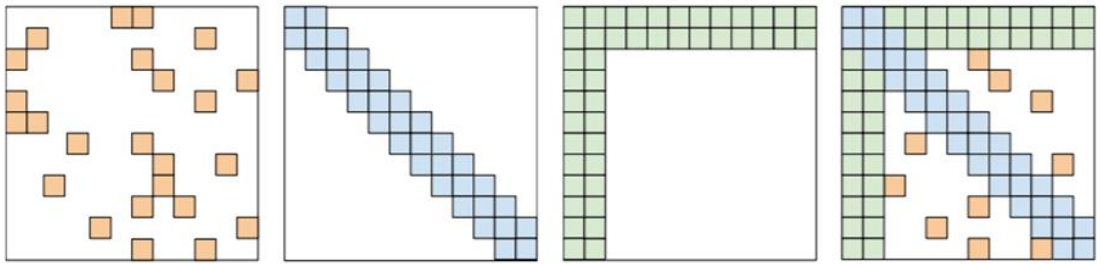
ETC Extended transformer construction (2020)

- Variant of Sparse Transformer
- It introduces a new **global-local attention mechanism**.
  - ✓ global-to-global (g2g),
  - ✓ global-to-local (g2l)
  - ✓ local-to-global (l2g)
  - ✓ local-to-local (l2l).
- Overall, ETC is quite similar to the Longformer in the way it introduces global auxiliary tokens. These tokens are trainable parameters and can be interpreted as a form of memory that pools across the sequence to collect global sequence information

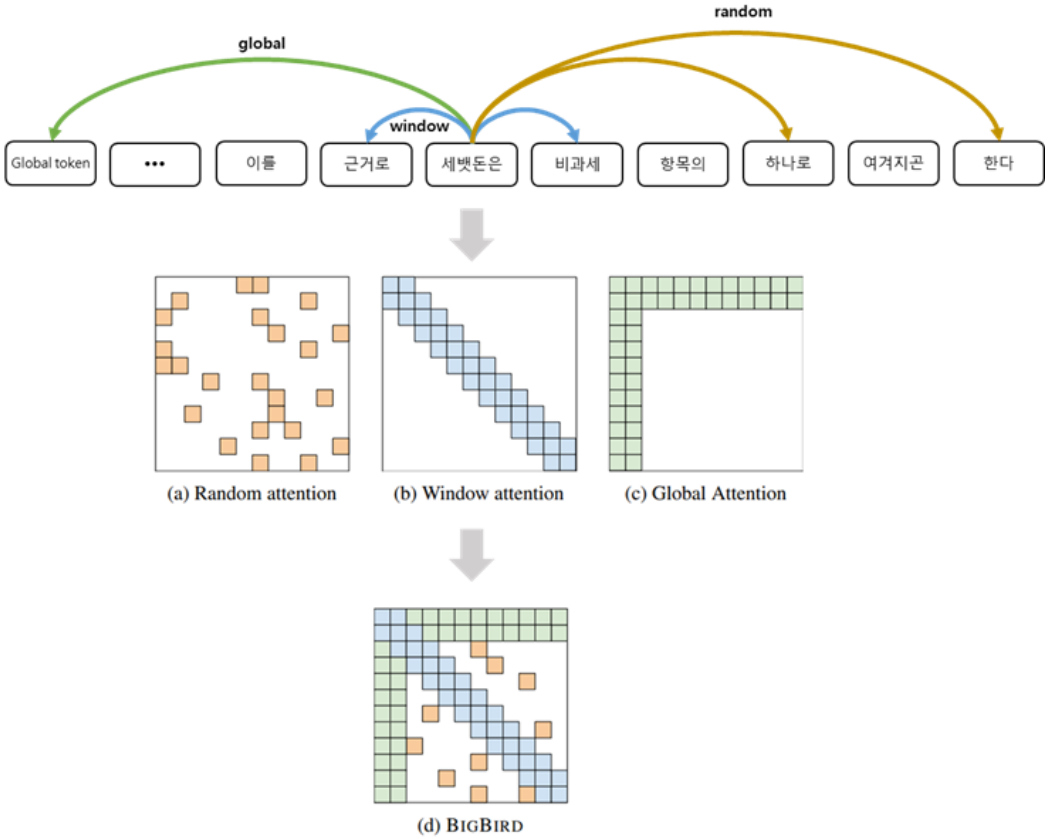
- The memory complexity of the ETC model **is** $\boldsymbol{O(n_g^2 + n_g N)}$
  - ✓ $\boldsymbol{n_g}$ is the number of global tokens
  - ✓N is the input sequence length.
- **Cannot be used for auto-regressive decoding.**
- This is because we are not able to compute causal masks because of the global attention

BigBird(2020)

- The BigBird model (Zaheer et al., 2020) is another Transformer for modeling longer sequences and is primarily built on top of ETC (Ainslie et al., 2020).
- Global Attention
  Fundamentally, the idea of using global memory can be traced all the way back to Longformer/ETC and Set Transformer model. Notably, the global memory in Big Bird is extended to contain tokens within the sequence, instead of simply parameterized memory. The authors call this the 'internal transformer construction (ITC)' in which a subset of indices is selected as global tokens. This can be interpreted as a memory based approach.
- Sliding Window Attention
  The window-ed attention was first proposed in early localbased attention models (Image Transformer, Compressed Attention and/or Sparse Transformer). In BigBird, each query attends to w/2 tokens to the left and w/2 tokens to the right. This corresponds to a fixed pattern (FP) approach.
- Random Attention
  Finally, each query attends to r random keys. This pattern is fixed.



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



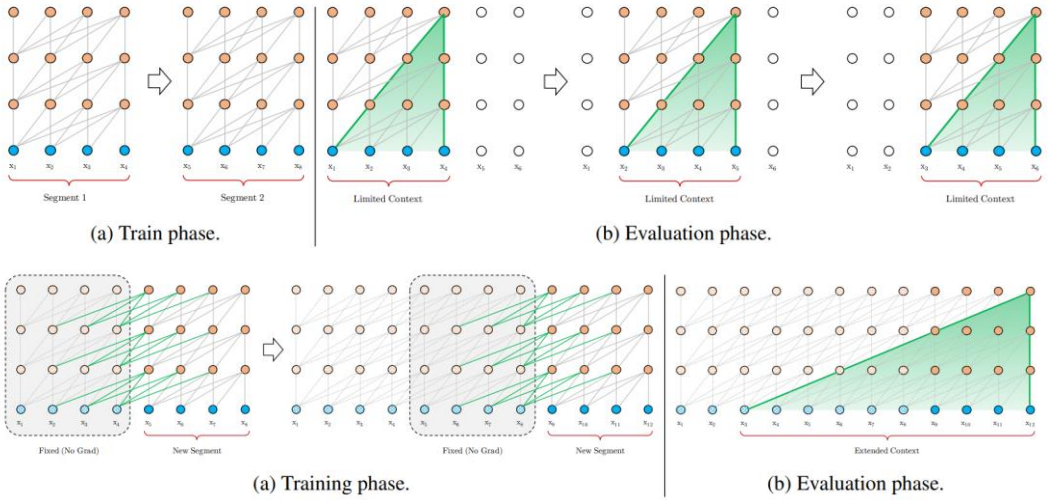
https://soundprovider.tistory.com/entry/2020-Big-Bird-Transformers-for-Longer-Sequences#i-%--Pretraining%--and%--MLM

Recurrence

Recurrence

- A natural extension to the **blockwise** method
- connect these blocks via recurrence.
- Transformer-XL (Dai et al., 2019) proposed a segment-level recurrence mechanism that connects multiple segments and blocks.

Transformer-XL Model



Low Rank/ Kernels

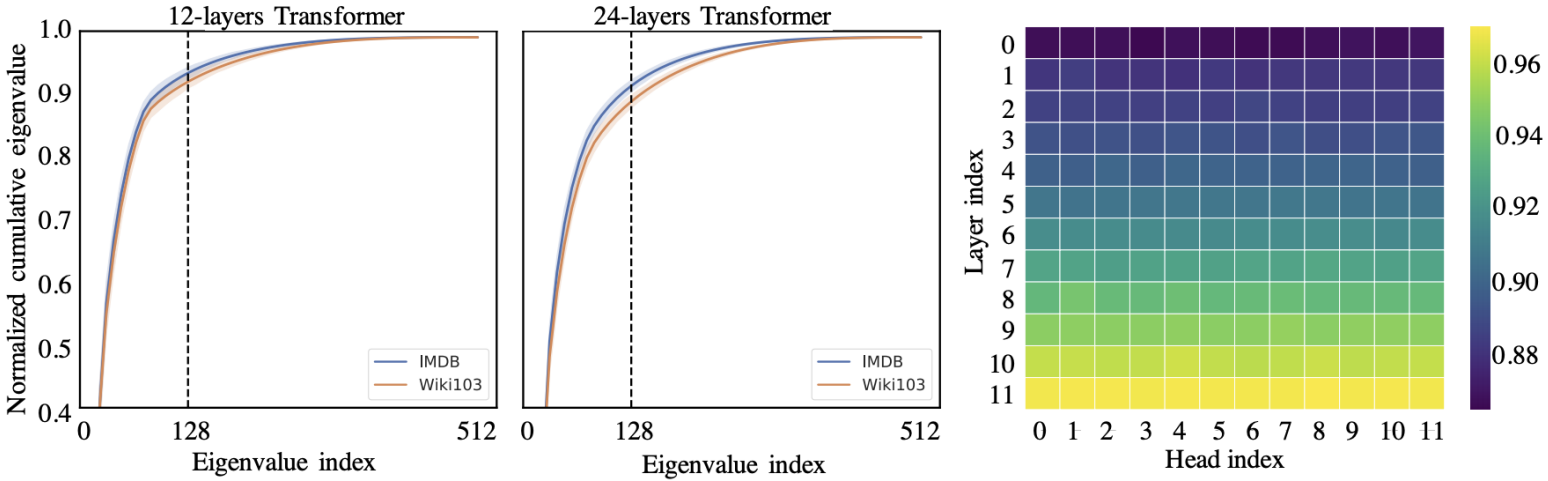
Low Rank Methods

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

- Linformer
- Synthesizer

Linformer(2020)

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) = \underbrace{\text{softmax}\left[\frac{QW_i^Q(KW_i^K)^T}{\sqrt{d_k}}\right]}_{P} VW_i^V$$

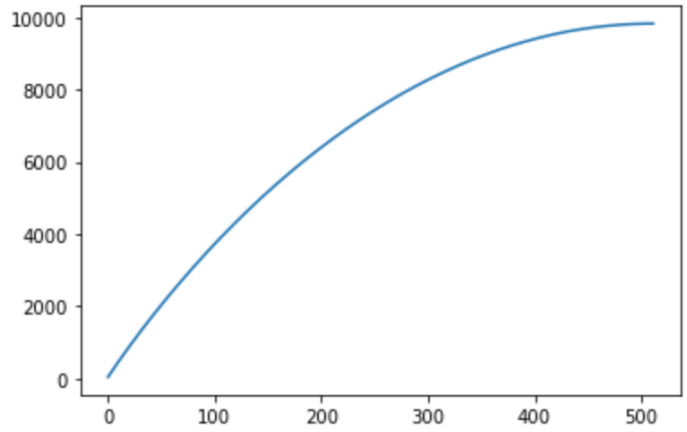


Linformer(2020)

```
m = np.random.normal(size=(512, 512))
```

```
plt.plot(np.cumsum(svd(m)))
```

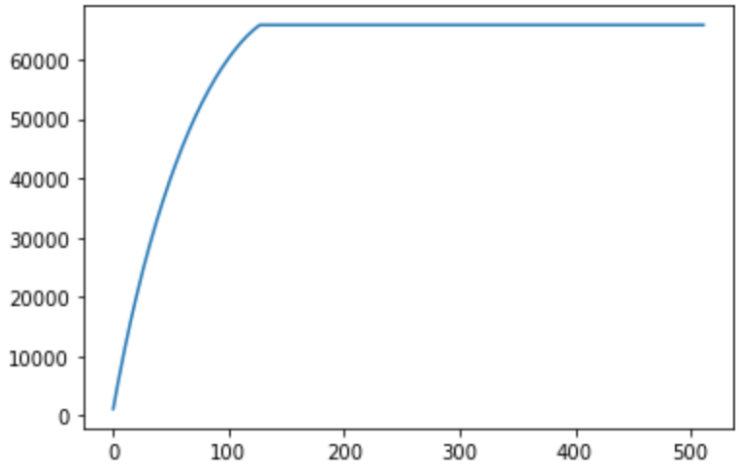
```
[<matplotlib.lines.Line2D at 0x7fba4d654940>]
```



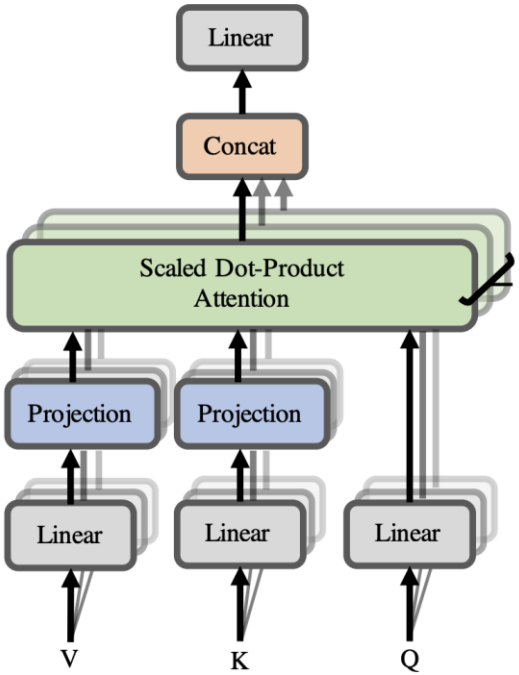

```
m = np.random.normal(size=(512, 512))
mt = m[:, :128]
```

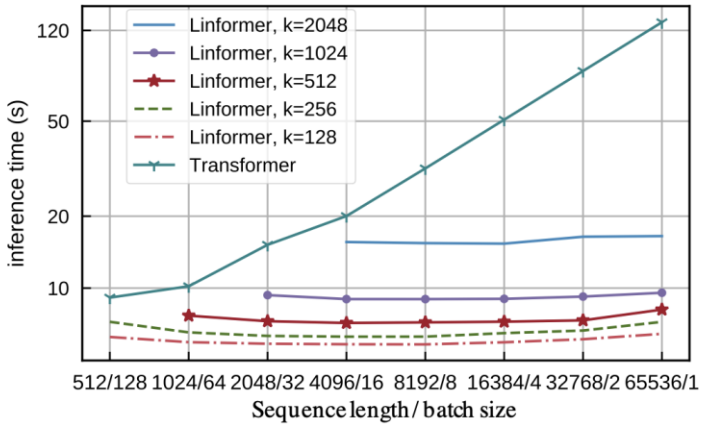
```
plt.plot(np.cumsum(svd(mt @ mt.T)))
```

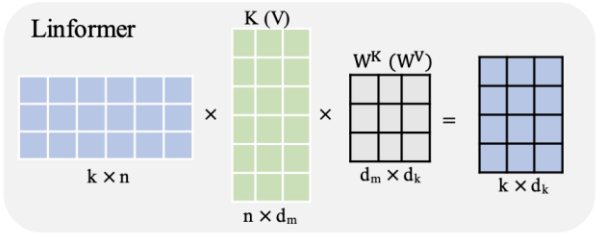
```
[<matplotlib.lines.Line2D at 0x7fba4d7f5f40>]
```



Linformer(2020)







Linformer(2020)

$$
\begin{aligned}
\overline{\text{head}_i} &= \text{Attention}(QW_i^Q, E_iKW_i^K, F_iVW_i^V) \\
&= \underbrace{\text{softmax}\left(\frac{QW_i^Q(E_iKW_i^K)^T}{\sqrt{d_k}}\right)}_{\bar{P}:n\times k} \cdot \underbrace{F_iVW_i^V}_{k\times d},
\end{aligned}
$$

Kernels

Another recently popular method to improve the efficiency of Transformers is to view the attention mechanism through kernelization. The usage of kernels enable clever mathematical re-writing of the self-attention mechanism to avoid explicitly computing the N X N matrix.

- Performer
- Linear Transformers

Linear Transformer(2020)

$$A_l(x) = V' = \text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)V.$$

$$V_i' = \frac{\sum_{j=1}^{N} \text{sim}\left(Q_i, K_j\right) V_j}{\sum_{j=1}^{N} \text{sim}\left(Q_i, K_j\right)}. \qquad \text{sim}\left(q, k\right) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$$

Linear Transformer(2020)

$$V_i' = \frac{\sum_{j=1}^{N} \mathrm{sim}\left(Q_i, K_j\right) V_j}{\sum_{j=1}^{N} \mathrm{sim}\left(Q_i, K_j\right)}. \qquad \mathrm{sim}(q,k) := \phi(q)^T \phi(k)$$

$$V_i' = \frac{\sum_{j=1}^{N} \phi\left(Q_i\right)^T \phi\left(K_j\right) V_j}{\sum_{j=1}^{N} \phi\left(Q_i\right)^T \phi\left(K_j\right)},$$

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

Linear Transformer(2020)

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

$$\phi(x) = \mathrm{elu}(x) + 1$$

Orthogonal Efficiency Efforts

Weight Sharing

Quantization/ Mixed Precision

Knowledge Distillation

Neural Architecture Search

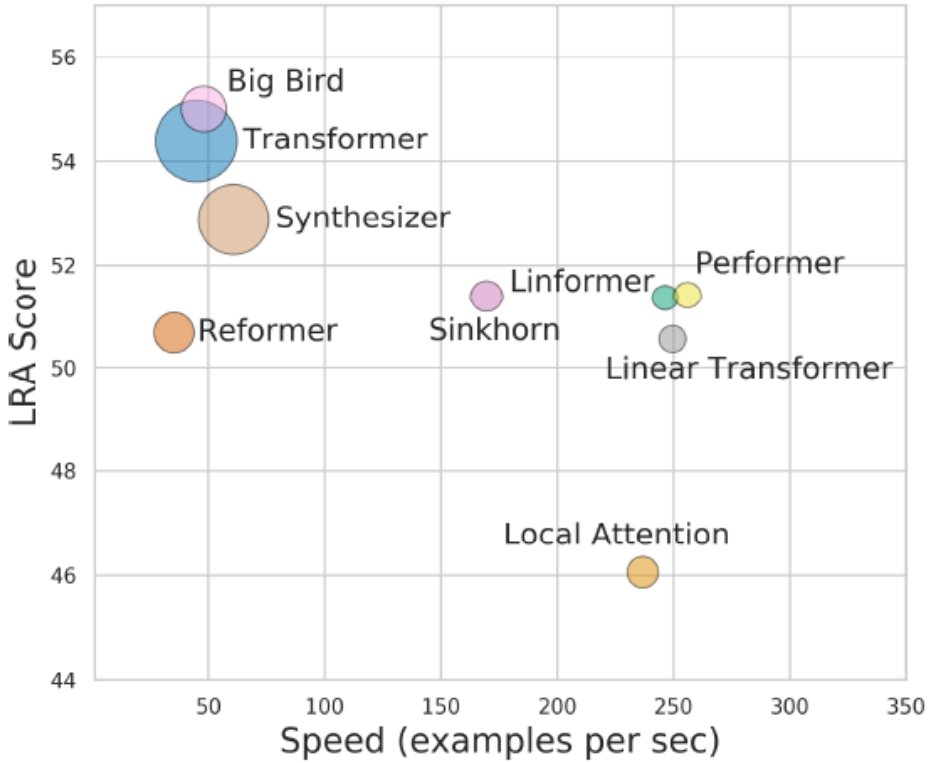
Task Adapters

Benchmark

LRA LONG RANGE ARENA: A BENCHMARK FOR EFFICIENT TRANSFORMERS (Tay et al., 2020)

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

- Task
  - LONG LISTOPS
  - BYTE-LEVEL TEXT CLASSIFICATION
  - BYTE-LEVEL DOCUMENT RETRIEVAL
  - IMAGE CLASSIFICATION ON SEQUENCES OF PIXELS
  - PATHFINDER (LONG-RANGE SPATIAL DEPENDENCY)
  - PATHFINDER-X (LONG-RANGE SPATIAL DEPENDENCIES WITH EXTREME LENGTHS)



Thanks