Performers

Author	Krzysztof Choromanski, Valerii Likhosherstov
Created time	@2021년 5월 26일 오전 6:09
Date	
Link	https://arxiv.org/pdf/2009.14794.pdf
Organization	Alan Turing Institute, DeepMind Google, Google, University of Cambridge
Presented @	ICLR 2021
Property	RETHINKING ATTENTION WITH PERFORMERS
Published	Mar 2021
Remarks	
Tags	Conference
Tags 1	
Tags 1 1	
Type	

- Performer code can be found in https://github.com/google-research/google-research/tree/master/performer
- Google AI Blog : https://ai.googleblog.com/2020/10/rethinking-attention-with-performers.html
- LightOn AI Meetup Video by the author, Krzysztof Choromanski. : https://youtu.be/NzlrQgb_KZ4

▼ Code for Transformer models on protein data can be found in

- https://github.com/google-research/google-research/tree/master/protein_lm
- Code for building and leveraging generative models on unlabeled protein data.
- Implements BERT and autoregressive models for proteins, as described in Biological Structure and Function Emerge from Scaling Unsupervised Learning to 250 Million Protein Sequences and ProGen: Language Modeling for Protein Generation.
- ProGen
    - Generative modeling for protein engineering is key to solving fundamental problems in synthetic biology, medicine, and material science.

▼ Rethinking attention with performer (paper summary)

Keywords: performer, transformer, attention, softmax, approximation, linear, bert, bidirectional, unidirectional, orthogonal, random, features, FAVOR, kernel, generalized, sparsity, reformer, linformer, protein, trembl, uniprot

Abstract

Performers : Transformer architectures which can estimate regular (softmax) full-rank-attention Transformers with provable accuracy, but using only linear (as opposed to quadratic) space and time complexity, without relying on any priors such as sparsity or low-rankness.

FAVOR+ : Performers use a novel Fast Attention Via positive Orthogonal Random features approach (FAVOR+) to approximate softmax attention-kernels, which may be of independent interest for scalable kernel methods.

- Performers는 sparsity나 low-rankness같은 것에 의존하지 않고 2차 공간일때와는 반대로 오직 선형 공간과 시간복잡성만을 사용해 입증가능한 정확도를 가지고 regular 또는 softmax full-rank-attention Transformers를 추정합니다.
- 또 Performers는 FAVOR+라는 mechanism을 만들어 사용했는데 이는 softmax attention kernels을 근사하기 위해 positive Orthogonal Random feature를 통한 빠른 Attention을 수행해 확장가능한 커널 방법으로 독립적으로 주목할 만한 새로

운 주제라고 이야기합니다. FAVOR+는 softmax를 넘어 kernel화 할 수 있는 attention mechanism을 효율적으로 모델링하는데 사용할 수 있다고 합니다.

- Transformers have huge memory and compute requirements because they construct an Attention matrix, which grows quadratically in the size of the input. The Performer is a model that uses random positive orthogonal features to construct an unbiased estimator to the Attention matrix and obtains an arbitrarily good approximation in linear time! The method generalizes beyond attention and opens the door to the next generation of deep learning architectures.
- New type of transformer models. Performers try to approximate the transformer without running into classic transformer bottleneck which is that the attention matrix in the transformer has space and compute requirements that are quadratic in the size of the input and that limits how much you can put into the models so it limits how long of text you can input of you work with text or how big your images are. FAVOR+ is potentially useful beyond transformers. It's apparently been here developed in the realm of the transformers but this can be independent interest for scalable kernel methods.

Limitations

Transformers rely on a trainable attention mechanism that identifies complex dependencies between the elements of each input sequence. Unfortunately, the regular Transformer scales quadratically with the number of tokens L in the input sequence, which is prohibitively expensive for large L and precludes its usage in settings with limited computational resources even for moderate values of L.

Several solutions have been proposed to address this issue. ... Those methods critically rely on kernels admitting explicit representations as dot-products of finite positive-feature vectors.

Unfortunately, there is a lack of rigorous guarantees for the representation power produced by such methods, and sometimes the validity of sparsity patterns can only be verified empirically through trial and error by constructing special GPU operations (e.g. either writing C++
CUDA kernels (Child et al., 2019) or using TVMs (Beltagy et al., 2020)).

- The regular Transformers는 입력 시퀀스의 토큰의 수 L에 따라 2차원으로 확장됩니다. 이는 L이 큰 값인 경우 그 비용이 엄청나게 비싸지고 심지어 중간값의

L에서도 제한된 계산 리소스가 있는 설정에서는 사용을 금하다 싶히 합니다.

- 이 문제를 해결하기 위한 다양한 시도들이 있었지만 대부분 additional constraints를 통합하는 방법으로 이러한 방법은 유한 양의 특성 벡터의 내적을 명시적 표현을 인정하는 커널에 심각하게 의존합니다.
- 기존 Transformer만큼의 representation 성능을 엄격하게 보장한다고는 할 수 없었고 또 때때로 희소성 패턴의 유효성은 특별한 GPU 작업을 구성하거나 TVM을 사용해 시행 착오를 통해 경험적으로만 검증 할 수 있었습니다.

Other techniques which aim to reduce Transformers' space complexity include reversible residual layers allowing one-time activation storage in training (Kitaev et al., 2020) and shared attention weights.

These constraints may impede application to long-sequence problems, where approximations of the attention mechanism are not sufficient

- 그 외에도 Transformer의 공간복잡도를 줄이기 위해 reversible residual network을 통해 학습과정에서 activation을 위해 저장고 어텐션의 shared 웨이트들을 공유하는 방법들도 있었습니다. 하지만 이러한 방법들은 어텐션에 대한 충분한 근사가 부족한 긴 시퀀스 문제에 대한 적용을 방해 할 수 있다.

Performers

Performers, capable of provably accurate and practical estimation of regular (softmax) full-rank attention, but of only linear space and time complexity and not relying on any priors such as sparsity or low-rankness.

Performers use the Fast Attention Via positive Orthogonal Random features (FAVOR+) mechanism, leveraging new methods for approximating softmax and Gaussian kernels, which we propose.

- Performer는 `full-rank attention`에 대해 정확하고 실질적으로 평가할 수 있는 모델이지만 선형적인 시간 및 공간 복잡도를 가지며, **sparsity**나 **low-rankness**등과 같은 어떠한 것에도 의존하지 않습니다.
- Performer는 FAVOR+를 이용합니다. FAVOR+는 저자들이 제안하는 가우시안 커널과 소프트맥스를 근사하기 위한 새로운 메소드입니다. **결과적으로 Performer는 커널을 이용해 기존의 트랜스포머와 함께 사용 가능한 선형구조로 강력한 이론적인 보증을 해주는 방법이기도 합니다.**

- FAVOR+는 소프트맥스를 넘어 다른 커널화 가능한 어텐션 메커니즘을 효율적으로 모델링하는 것에 적용할 수 있습니다. FAVOR+의 표현력은 처음으로 대규모 스케일의 태스크들에 대해 다른 커널들과 소프트맥스를 정확하게 비교하고 최적의 어텐션-커널들을 찾는데 중요한 역할을 합니다. FAVOR+는 기존의 어텐션을 확장 가능한 대체방법으로써 트랜스포머를 넘어 다양한 분야에 적용될 수 있습니다. 적용 가능한 분야로는 컴퓨터비전, 강화학습, 소프트맥스 크로스엔트로피 로스를 이용한 학습, Combinatorial 옵티마이제이션과 같은 다양한 곳에 적용될 수 있습니다.

FAVOR+는 reversible layer와 cluster-based attention등과도 결합될수도 있습니다.

The performer, linear attention using Kernel tricks.

- 작년 2020년에 구글에서 나와서 ICLR 2021 oral session으로 채택된 페이퍼이다.

▼ 논문에서는 encoder 위주의 설명이 있음. All the improvement demonstrated by the performer encoder.

  - Transformer 모델들은 대부분 encoder 모델을 decoder 자리에도 사용한다.
  - 윤철희박사님 논문에서 위와같은 설명도 있었다. sum to 1만되면 된다. operator 라고 하면 distribution 을 뽑음. 트릭 스페이스에서 디스트리뷰션 스페이스로 unbounded space에서

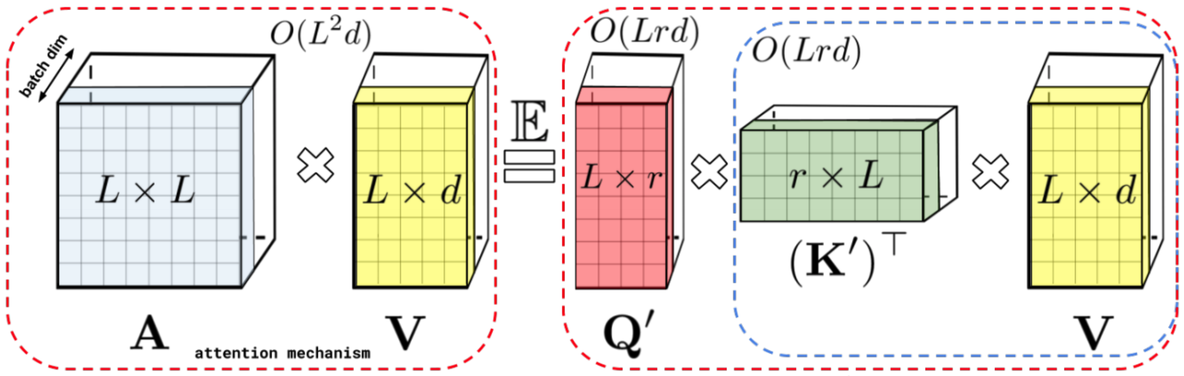
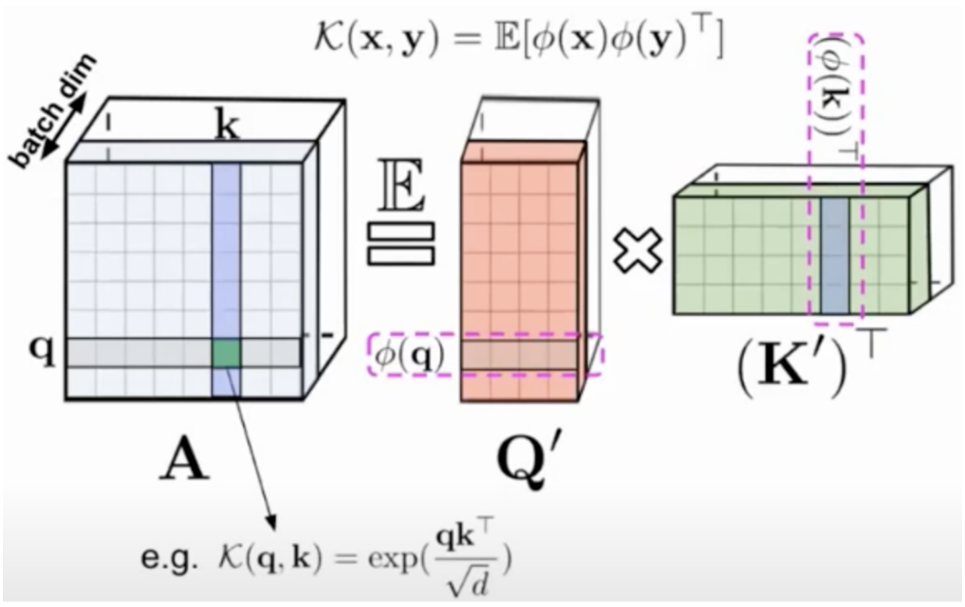


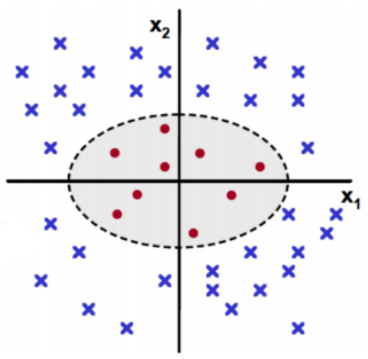
Figure 1: Approximation of the regular attention mechanism **AV** (before $\mathbf{D}^{-1}$-renormalization) via (random) feature maps. Dashed-blocks indicate order of computation with corresponding time complexities attached.

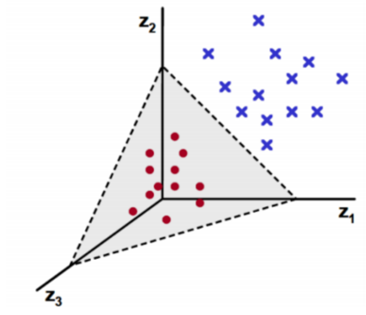

$\mathcal{K}(\mathbf{x},\mathbf{y}) = \mathbb{E}[\phi(\mathbf{x})\phi(\mathbf{y})^\top]$
batch dim
$\mathbf{k}$
$\mathbf{q}$
$\mathbf{A}$
$\mathbb{E}$
$=$
$\phi(\mathbf{q})$
$\mathbf{Q}'$
$\phi(\mathbf{k})^\top$
$(\mathbf{K}')^\top$
e.g. $\mathcal{K}(\mathbf{q},\mathbf{k}) = \exp(\frac{\mathbf{q}\mathbf{k}^\top}{\sqrt{d}})$

Recab

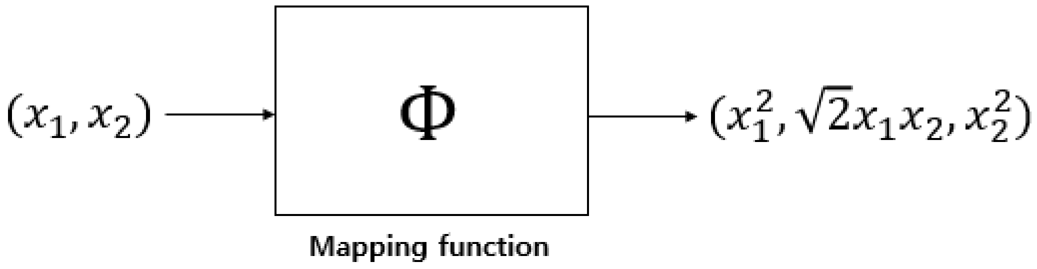
▼ recab

- **Kernel**





example of Polynomial Kernel



exampe of SVM

$$\text{maximize} \quad f(c) = \sum_{n}^{N} c_n - \frac{1}{2} \sum_{i}^{N} \sum_{j}^{N} c_i c_j y_i y_j (x_i^\top x_j)$$

$$\text{maximize} \quad f(c) = \sum_{n}^{N} c_n - \frac{1}{2} \sum_{i}^{N} \sum_{j}^{N} c_i c_j y_i y_j (\varphi(x_i)^\top \varphi(x_j))$$

- **Kernel Trick**

$$\text{maximize} \quad f(c) = \sum_{n}^{N} c_n - \frac{1}{2} \sum_{i}^{N} \sum_{j}^{N} c_i c_j y_i y_j K(x_i, x_j)$$

doing the dot-product in lower dimensional space, but still get the same result as performing the dot-product after the projection.

- **RBF(Radial Basis Function) kernel (Gaussian kernel)**

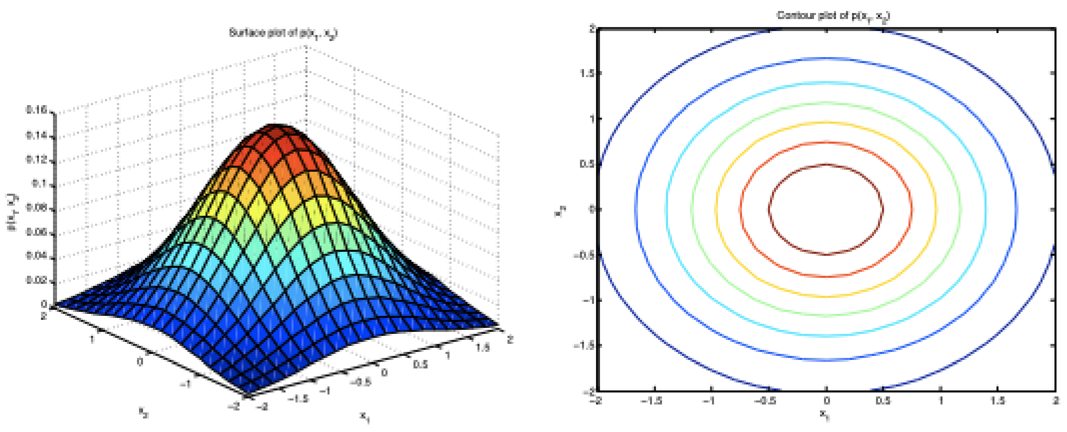
Gaussian kernel

$$k_\sigma(\mathbf{x},\mathbf{x}') = \exp\left(-\frac{||\mathbf{x}-\mathbf{x}'||^2}{2\sigma^2}\right) = \exp\left(-\frac{\mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T\mathbf{x}' + \mathbf{x}'^T\mathbf{x}'}{2\sigma^2}\right)$$

- Depends on a *width* parameter σ
- The smaller the width, the more prediction on a point only depends on its nearest neighbours
- Example of *Universal* kernel: they can uniformly approximate any arbitrary continuous target function (pb of number of training examples and choice of σ)

- **Spherical Gaussian**

확률 분포가 구면(원형) 대칭, 구면 가우시안(Spherical Gaussian)



(a) Spherical Gaussian (diagonal covariance, equal variances)

Random Features

random feature map *z* 를 통해 *K* 에 근사; *z* : *R_L*↦*R_R*

$$\varphi(x_i)^T\varphi(x_j) \approx z(x_i)^T z(x_j)$$

Performers

FAVOR+ Attention

Finding the Softmax kernel through the Gaussian kernel

$$\mathrm{K}(\mathbf{x},\mathbf{y}) = \phi(\mathbf{x})^\top \phi(\mathbf{y})$$

$$\mathbf{A}(i,j) = \mathrm{K}(\mathbf{q}_i,\mathbf{k}_j) = \exp(\mathbf{q}_i\mathbf{k}_j^\top) = \phi(\mathbf{q}_i)^\top \phi(\mathbf{k}_j)$$

$$\phi(\mathbf{x})_{\mathrm{SM}} = \frac{1}{\sqrt{m}}\exp(-\frac{||\mathbf{x}||^2}{2})(\exp(w_1^\top\mathbf{x}),\ldots,\exp(w_m^\top\mathbf{x})))$$

$$\mathbf{A}(i,j) = \mathrm{K}(\mathbf{q}i,\mathbf{k}_j) = \exp(\mathbf{q}_i\mathbf{k}_j^\top) = \phi_{\mathrm{SM}}(\mathbf{q}_i)^\top \phi_{\mathrm{SM}}(\mathbf{k}_j)$$

Softmax Kernel

$$A = \exp\left(\frac{Q}{\sqrt[4]{d}}\left(\frac{K}{\sqrt[4]{d}}\right)^\top\right)$$

$$A = \exp(QK^\top)$$

Attention 식 (simple version_

$$K_{\mathrm{softmax}}(x_i, x_j) = \exp(x_i^\top x_j)$$

Softmax Kernel definition

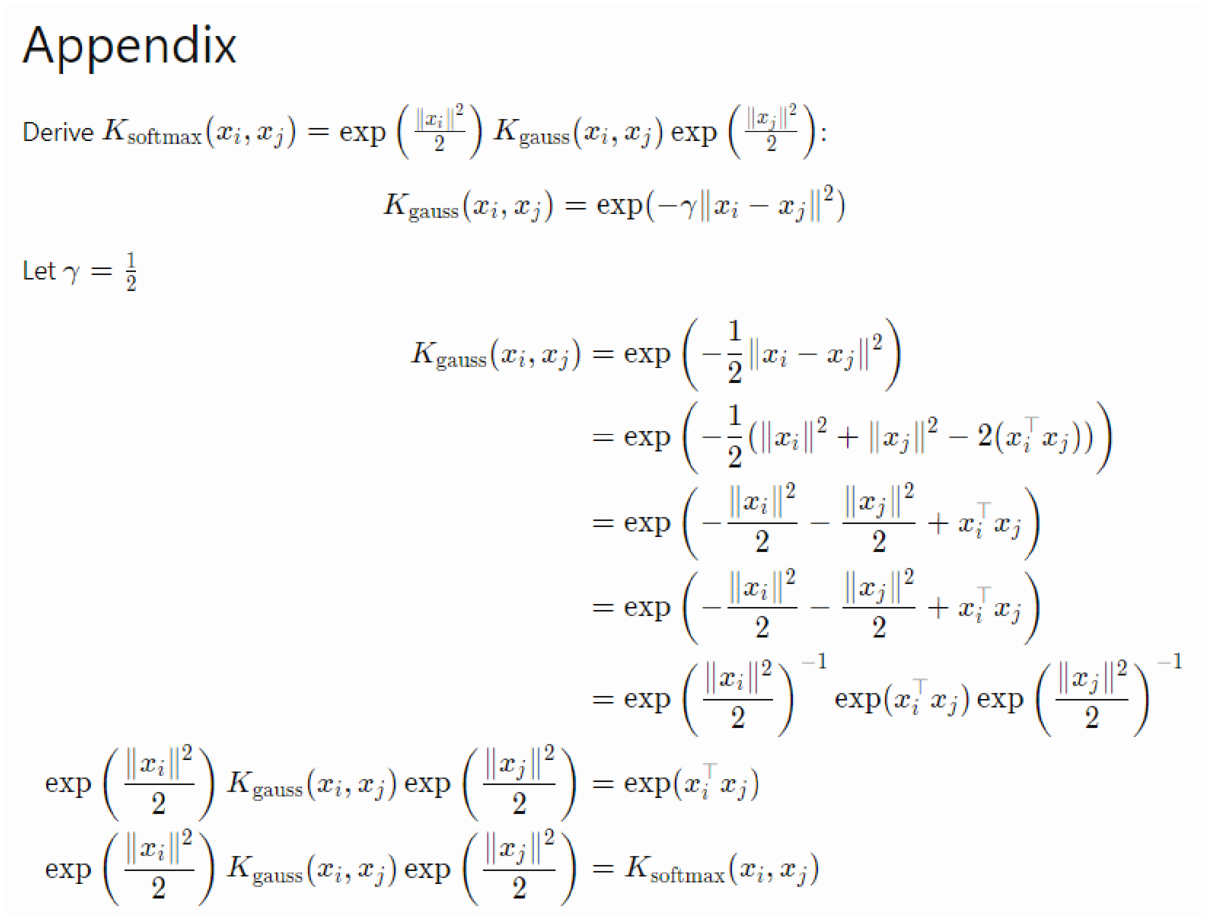
$$A(i,j) = K_{\mathrm{softmax}}(q_i^\top, k_j^\top)$$

Softmax Kernel을 사용해 Attention 식을 다시 씀.

$$K_{softmax}(x_i, x_j) \approx z(x_i)^\top z(x_j)$$

더 낮은 차원으로 근사

https://teddykoker.com/2020/11/performers/



Appendix

Derive $K_{\text{softmax}}(x_i, x_j) = \exp\left(\frac{\|x_i\|^2}{2}\right) K_{\text{gauss}}(x_i, x_j) \exp\left(\frac{\|x_j\|^2}{2}\right)$:

$$K_{\text{gauss}}(x_i, x_j) = \exp(-\gamma\|x_i - x_j\|^2)$$

Let $\gamma = \frac{1}{2}$

$$\begin{aligned}
K_{\text{gauss}}(x_i, x_j) &= \exp\left(-\frac{1}{2}\|x_i - x_j\|^2\right) \\
&= \exp\left(-\frac{1}{2}(\|x_i\|^2 + \|x_j\|^2 - 2(x_i^\top x_j))\right) \\
&= \exp\left(-\frac{\|x_i\|^2}{2} - \frac{\|x_j\|^2}{2} + x_i^\top x_j\right) \\
&= \exp\left(-\frac{\|x_i\|^2}{2} - \frac{\|x_j\|^2}{2} + x_i^\top x_j\right) \\
&= \exp\left(\frac{\|x_i\|^2}{2}\right)^{-1} \exp(x_i^\top x_j) \exp\left(\frac{\|x_j\|^2}{2}\right)^{-1}
\end{aligned}$$

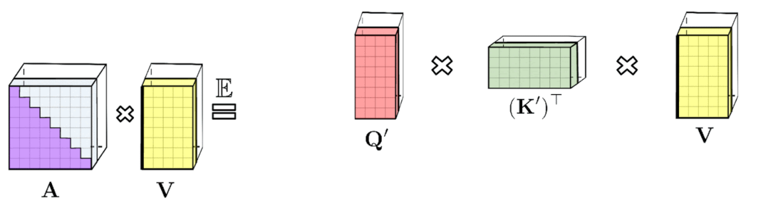
$$\exp\left(\frac{\|x_i\|^2}{2}\right) K_{\text{gauss}}(x_i, x_j) \exp\left(\frac{\|x_j\|^2}{2}\right) = \exp(x_i^\top x_j)$$

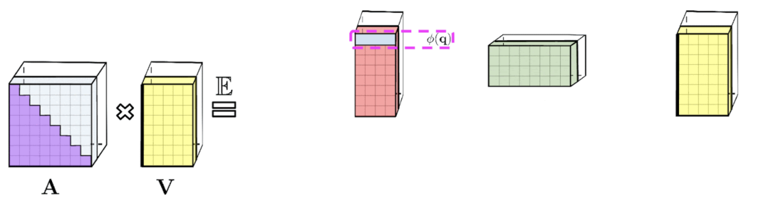
$$\exp\left(\frac{\|x_i\|^2}{2}\right) K_{\text{gauss}}(x_i, x_j) \exp\left(\frac{\|x_j\|^2}{2}\right) = K_{\text{softmax}}(x_i, x_j)$$

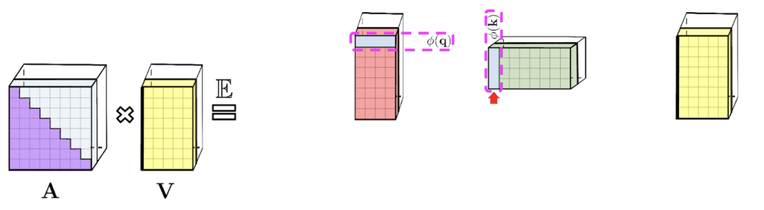
Illustrated Performers

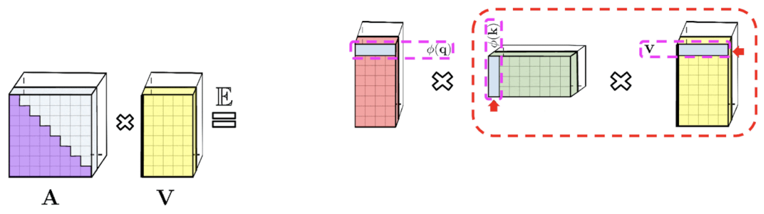
Left: Standard unidirectional attention requires masking the attention matrix to obtain its lower-triangular part.

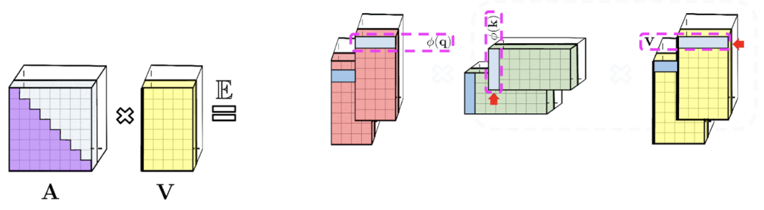
Right: Unbiased approximation on the LHS can be obtained via a prefix-sum mechanism, where the prefix-sum of the outer-products of random feature maps for keys and value vectors is built on the fly and left-multiplied by query random feature vector to obtain the new row in the resulting matrix.

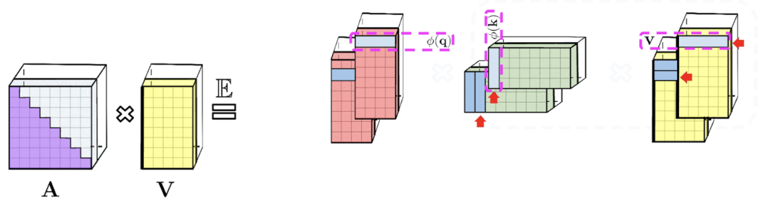

A
V
$\mathbb{E}$
$\mathbf{Q}'$
$(\mathbf{K}')^\top$
V

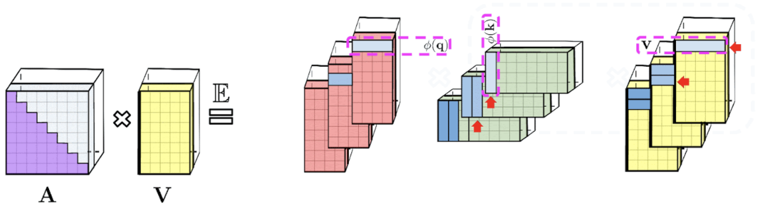

A
V
$\mathbb{E}$
$\phi(\mathbf{q})$

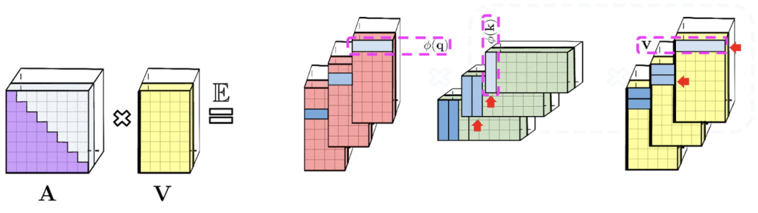

A
V
$\mathbb{E}$
$\phi(\mathbf{q})$
$\phi(\mathbf{k})$


A
V
$\mathbb{E}$
$\phi(\mathbf{q})$
$\phi(\mathbf{k})$
$\mathbf{v}$

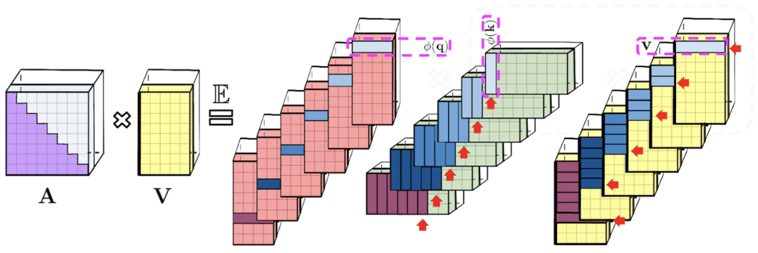

A
V
$\mathbb{E}$
$\phi(\mathbf{q})$
$\phi(\mathbf{k})$
$\mathbf{v}$


A
V
$\mathbb{E}$
$\phi(\mathbf{q})$
$\phi(\mathbf{k})$
$\mathbf{v}$





...x ***L***



Standard unidirectional attention 을 근사하기 위해 prefix-sum mechanism를 사용.

1. 먼저 Key와 Value 벡터들의 random feature maps끼리 outer-product를 on-the-fly로 계산한다.
2. 그 다음 왼쪽으로부터 Query의 random feature vector를 곱해줌으로써 다음에 올 행/token을 찾을 수 있다.
3. 찾아진 토큰으로부터 새로운 K와 V를 얻고 위의 과정 반복.