

# RetNet → Retentive Network

## Transformer

- RNN (recurrent neural network)의 단점

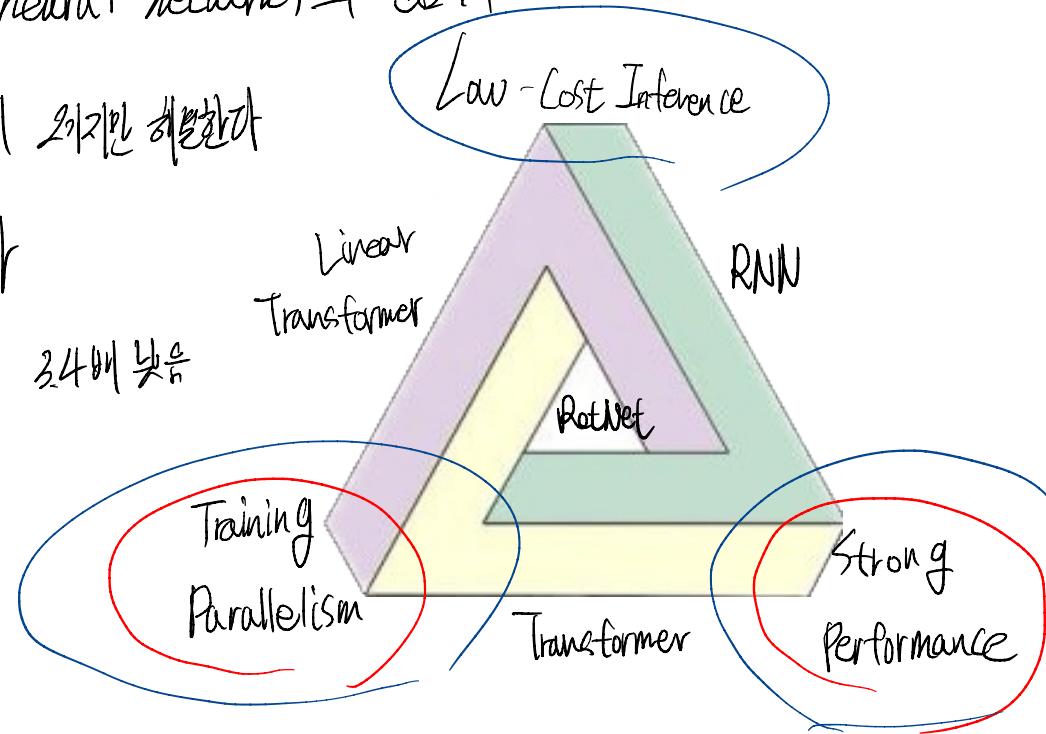
- 여러 문장의 상관성이 유지된다

- RetNet은 가능하다

- Transformer의 계산량은 3.4배 높음

- 8.4배의 높은 처리량

- 15.6배로 높은 계산시간



# Motivation

## RetNet

- Training Parallelism
- Inference Cost  $O(1)$
- Memory Complexity  $O(N)$

	Training Parallelism	Inference Cost	Memory Complexity	Performance
RNNs	✗	$O(1)$	$O(N)$	⬇️
Transformers	✓	$O(N)$	$O(N^2)$	⬆️
RetNet	✓	$O(1)$	$O(N)$	⬆️

Fig 2: RetNet achieves training parallelization, constant inference cost, linear long-sequence memory complexity, and good performance

# Training Parallelism

- RNN은 시퀀스를 반복 순차적으로 학습하여 훈련 속도가 매우 느림
- Transformer는 병렬 학습이 가능하지만 inference의 문제가 생김(추후 설명)
- RetNet은 단위, 반복, chunk-wise inference 다 가능  
(Self-attention 병렬화의 수용)

# Inference Cost + Memory Complexity

## Inference Cost (per time step)

- \* GPU memory

- \* throughput and latency (밀도 및 애플리케이션)

## Memory Complexity

- \* 시퀀스 길이에 따른 메모리 용량 확장성

RNN  $\rightarrow$  행렬 곱셈 (matrix multiplication), ~~인코딩~~:  $O(\text{길이} \times \text{길이} \times O(1))$   
인코딩 복잡성은 시퀀스 길이에 따라 ~~제한적~~ 증가

Transformer  $\rightarrow$  Self-attention (Inference Cost:  $O(N)$ )에서 ~~병행적으로~~ 증가  
memory-complexity  $O(N)$   $\Rightarrow$  Inference  $\sim O(N^2)$

RetNet : Transformer의 Self-attention 대비 (更快, 더 빠름)

→ Self-attention은 RNN과 함께 Self-attention 대비

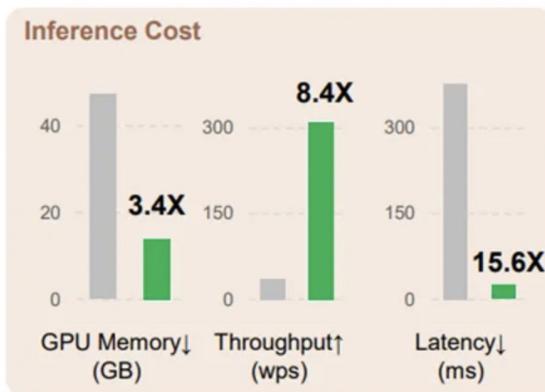


Fig 3: RetNet achieves orders of magnitude better inference cost than transformers. Results of inference cost are reported with 8k as input length (image from original paper)

## Performance

- self-attention 주변의 Gradient - Vanishing梯度
- Transformer가 충돌하기 쉬운 경우의 예

## RetNet: Overview

1. multi-head - attention  $\rightarrow$  multi scale retention  
 $d_{\text{model}}$
2. train & inference의 종합한 시스템 디자인의 사용  
Transformer의 비슷한 3가지 차는 병용 적용
  - a. Parallel representation은 training Parallel을 강화하여 GPU를 효율 사용
  - b. Recurrent representation은 메모리 제한 축면에서 O(1) Inference를 가능케 한다  
 $\rightarrow$  deployment cost & latency 개선, key-value Cache trick  $\frac{O(1)}{O(n)}$
  - c. chunk-wise RNN은 효율적으로 전문가용 모델링 가능 계산속도를 위해 각 쿠션 블록을 병렬로 연산하는 동시에 GPU 메모리를 충족하기 위한 GPU 메모리와 임계블록을 병렬적으로 연산

RetNet VS transformer

RetNet = RNN( $\text{avg}$ ) + transformer( $\text{avg}$ )

- RNN  $\hookrightarrow$  + Inference  $\hookrightarrow$  retention ( $\text{avg}$ )  $\xrightarrow{\text{mult}}$

영역  $\rightarrow$   $\text{avg}$  + RNN  $\hookrightarrow$  영역의 recurrent paradigm

# STEP 1. Parallel Representations for training

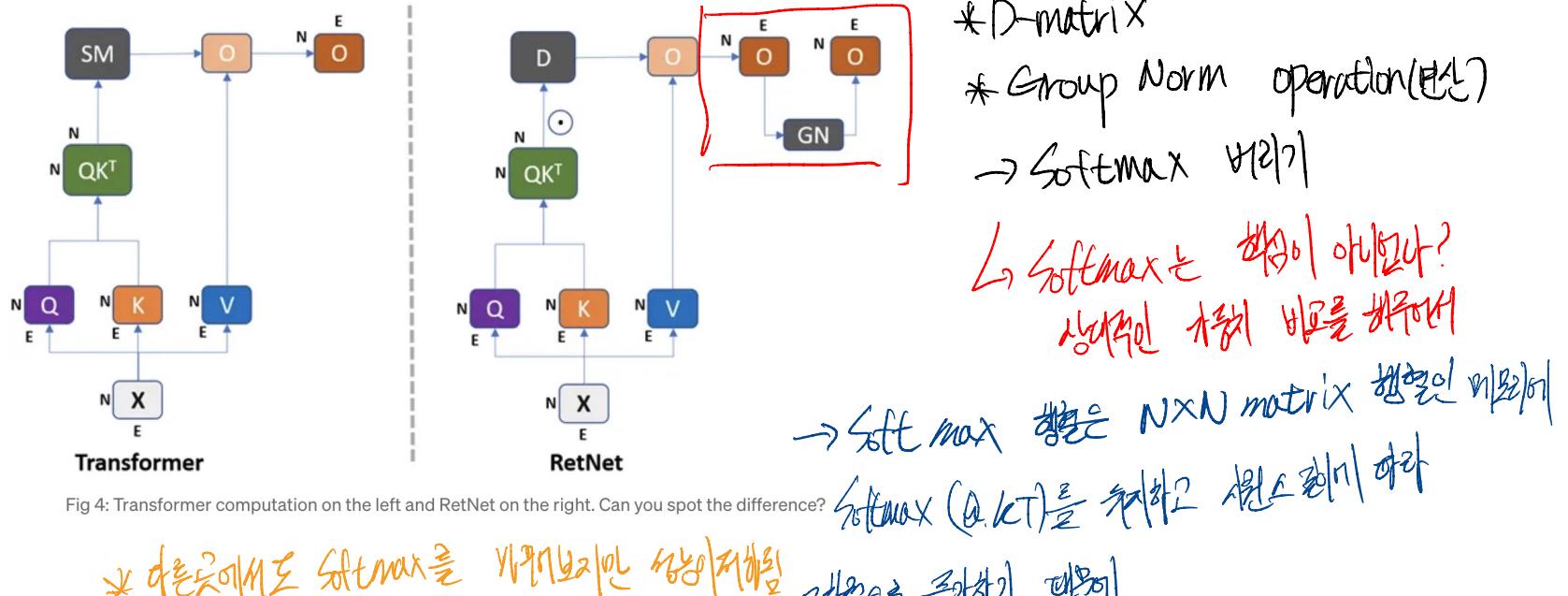


Fig 4: Transformer computation on the left and RetNet on the right. Can you spot the difference?

\* 다른곳에서도 Softmax를 사용하지만 성능 저하로  
계속으로 증가하기 때문

\* D-matrix와 Group Norm을 해온

Transformer의 Inference Time은  
적은 편이!

D-matrix와 Group Norm이 무엇이며 어떻게 흐름을 하는가

## Transformer에서 Softmax의 역할

### 1. 서로 다른 시간 단계에 차르게 가중치 적용

· 올바른 시간을 흐름하는지 흐름을 줌

→ 제한된 D-matrix는 올바른 시간을 흐름하는지 흐름을 주지만 그 외 모양이나 시각적인 차별화로 같은 Causal-mask

→ 각 시간의 단계가 향후 단계에 주는 가치는 것을 방지하는 동시에 모든 시간에 상대적으로 가중치 부여

→ D-matrix는 차운 단계가 초기 단계보다 중요하고 생각하므로 exponential decay weight 적용

→ Softmax는 시간의 단계에 가치를 부여함으로 흐름되지만

D-matrix는 exponential decay로 정확한 방식으로 모든 단계에 가치를 부여함

→ Softmax의 차원

· 정확한 효율적인 O(1) inference와 O(N)의 메모리 부하

## 2. Non-linearity (비선형성)

기울기와 편향이 변화하는 비선형

Softmax가 ~~아닌~~ QKT 주입은 레이어마다 성능이 향상되는 것을 제한하는 affine 변환이다

GroupNorm 비선형성을 조성하는데 와 GroupNorm을 사용하지 않거나 있다

The new Retention mechanism

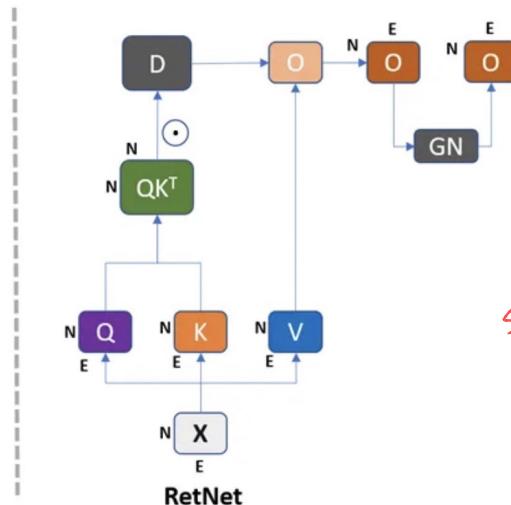
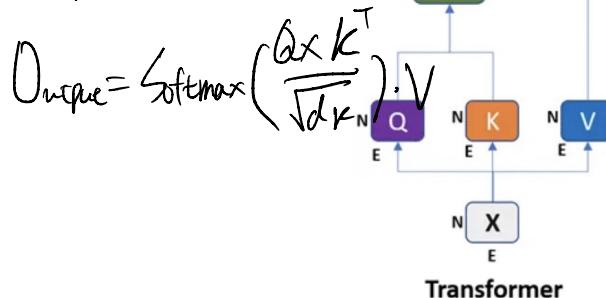
Retention = Recurrent + Self-attention

Transformer

$$Q = XW_Q,$$

$$K = XW_K,$$

$$V = XW_V$$



1) new input을 기반으로 하는  
recurrent Retention block 형태의  
 $O_n = \sum_{m=1}^h (Q_n \cdot A^{n,m} \cdot K_m^T) \cdot V_m$

Pos

Softmax가 위치정보를 통해 계산됨  
Posit 정보는 각 단계에 따라  
부이 아래와 같이 함

$$\text{Retention}(x) = \sum_{n=1}^N \underbrace{(Q_n (xe^{ie})^n)}_{\text{Pos}} \underbrace{(K_m (xe^{ie})^m)^T}_{\text{Pos}} \cdot V_m$$

Fig 4: Transformer computation on the left and RetNet on the right. Can you spot the difference?

$$\text{Retention}(x) = \sum_{m=1}^n \underbrace{(Q_m(xe^{ie}))}_{\text{Pos}} \underbrace{(K_m(xe^{ie})^T)}_{\text{Pos}} \cdot V_m$$

$\text{Pos}$ 은  $\text{Pos}$ 의 결과값,  $i$ 를 실수값으로 사용하여 헤드 단위로 하면  $i$ 를 정지 헤드로 한다.

$$Q = XW_Q \odot \Theta, \quad K = XW_K \odot \bar{\Theta}, \quad V = XW_V$$

$$\text{where}, \quad \Theta = e^{\frac{i}{d} \text{diag}(D_{nm})}, \quad D_{nm} = \begin{cases} \theta^{n-m}, & n \leq m \\ 0, & n > m \end{cases}$$

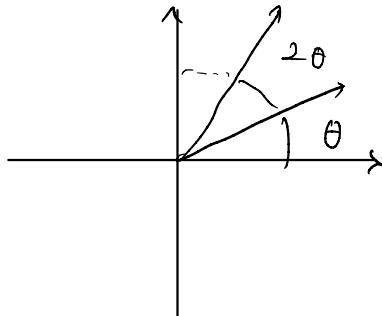
$$\text{Retention}(x) = [(QK)^T \odot D] \cdot V$$

→ Transformer는 주제 영역이 다른 매우 유사한 문장들이 가깝다

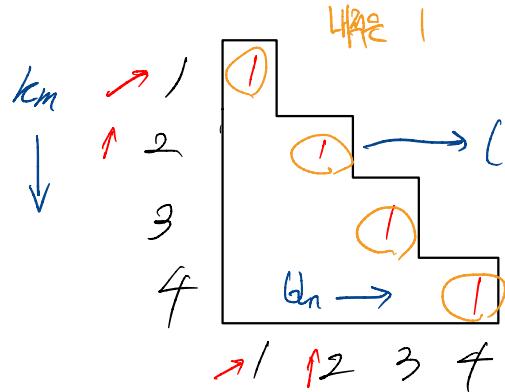
# Relative position Embedding (pos/pos')

Euler's formula

$$e^{i\theta} = \cos\theta + i\sin\theta$$



Retention의 θ는 벡터회전을 통해 Q 및 K행렬의 벡터회전을 통해 Q 및 K행렬에 상대적 위치정보를 인코딩한다  
벡터와 각각의 위치별 벡터 회전간의 Hadamard 곱을 통해서 학성



$$(Q_n \cdot k_m^T \cdot \Theta) \rightarrow 0$$

↳ when orthogonal

각각의  $Q_n, k_m$  벡터는 랜덤으로 초기화되어  
이해하기 편리하게  $n=m=1$  일 때  $e^{i\theta}/e^{im\theta}$ 가  
직선회전을 일으킨다

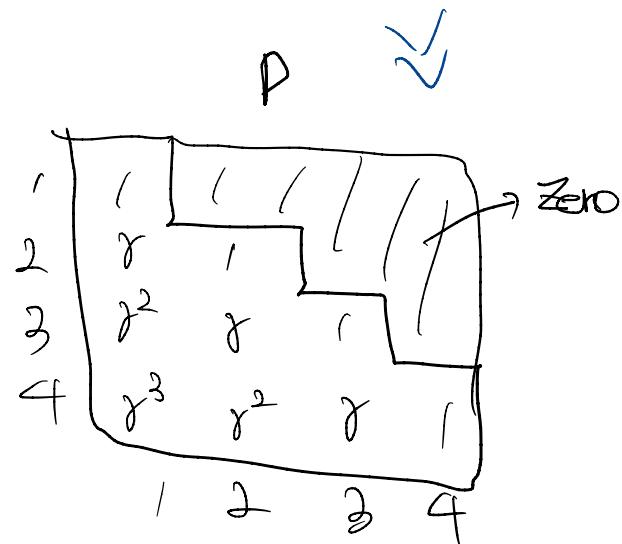
## Causal masking and exponential decay (P)

D-matrix는 과거와 현재에 대한 exponential decay weight 차이로 인해 Causal mask ~~effector~~

$D_{nm} \begin{cases} \gamma^{n-m}, & n \leq m \\ 0, & n > m \end{cases}$  exp-decay  $\rightarrow$  Softmax  $\rightarrow$  ( $Q[k]$ )의 빠른  $\gamma$ 의 exponentially decaying factor  $\Rightarrow$  가중치가 높아짐

$\hookrightarrow$  0을 출력하여 시퀀스 처리 과정에 적용한지 확인

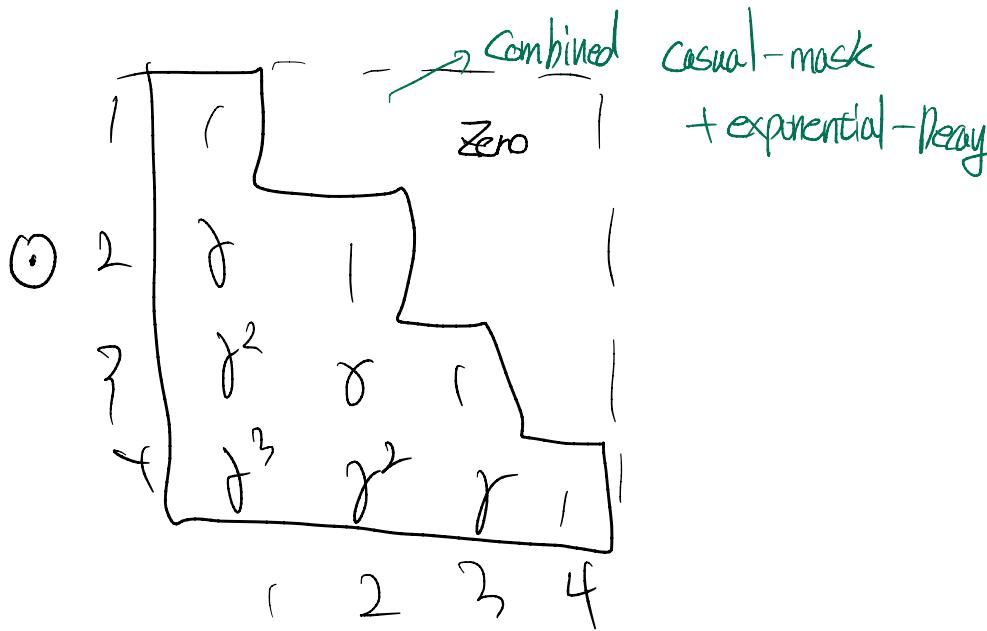
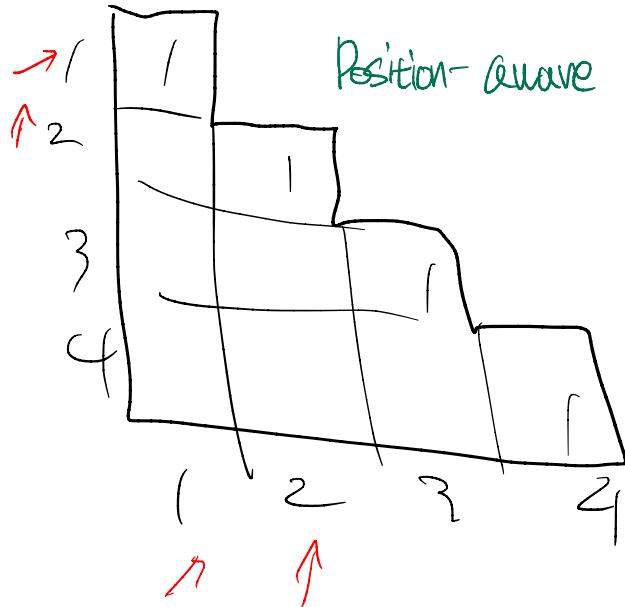
$\rightarrow$  Softmax 뒤 처리하고 히트맵에  
언어만 잘 적용된가



Position-aware ( $\theta, KV$ ) 와 D-매트릭스가 어떻게 결합하여  $X$ 에 있는 각 입력들은 그 최종적인 임베딩을 계산하는지 확인

Putting them together

→ 그림들에 차례대로 이용하여 병렬연산의 결합성을 한다



# Parallel training - working example

$$\text{입력 } \Rightarrow D=3 \quad N=2$$

$$Q = \begin{bmatrix} 1 & 2 & 1 \\ 3 & 2 & 3 \end{bmatrix} \quad k = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, V = \begin{bmatrix} 5 & 4 & 3 \\ 2 & 1 & 0 \end{bmatrix}$$

Step 1  $Qk^T$

$$Q.k^T = \begin{bmatrix} 1 & 2 & 1 \\ 3 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} = \begin{bmatrix} 8 & 20 \\ 16 & 40 \end{bmatrix}$$

Step 2 Hadamard product between  $Qk^T$  and  $D$

$$Qk^T \odot D = \begin{bmatrix} 8 & 20 \\ 16 & 40 \end{bmatrix} \odot \begin{bmatrix} 1 & 0 \\ 0 & 25 & 1 \end{bmatrix} = \begin{bmatrix} 8 & 0 \\ 4 & 40 \end{bmatrix}$$

Step 3 multiplying  $(Q \cdot K^T \text{ and } D)$  with  $V$

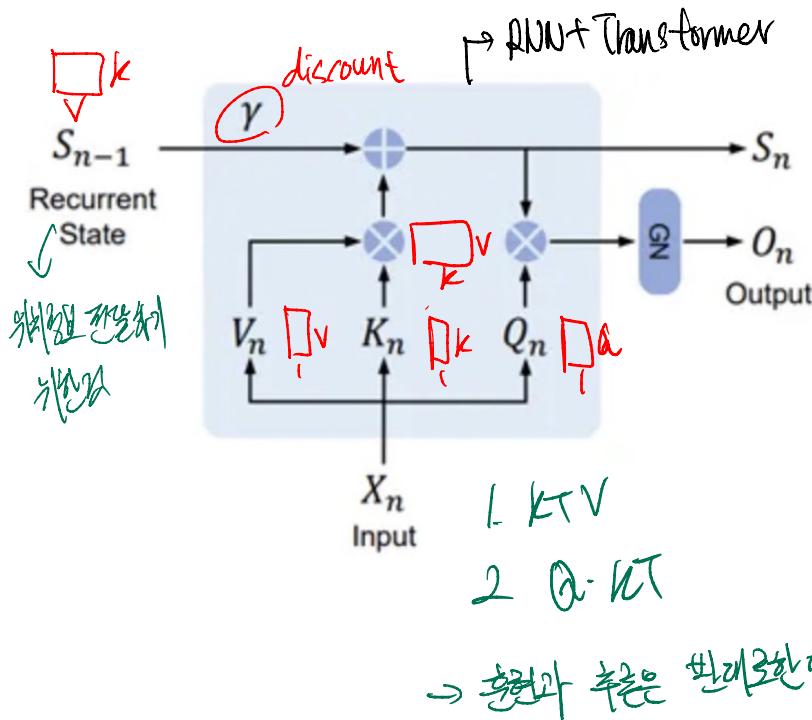
$$(Q \cdot K^T \cdot D) V = \begin{bmatrix} 80 \\ 40 \end{bmatrix} \begin{bmatrix} 5 & 0 & 0 \\ 2 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 40 & 32 & 24 \\ 100 & 56 & 12 \end{bmatrix}$$

$+0$

$4 \times 5 + 4 \times 2 \quad 4 \times 4 + 40 \times 1$

# Recurrent Retention for inference

$n | \theta_2$  복잡성 일부를 별별 계산함



$$S_n = \gamma S_{n-1} + k_n^T V$$

$$\text{Retention}(x_n) = \alpha_n S_n ; n=1,2,\dots (x)$$

$Q^T V$  대신  $K^T \cdot V$  → 되는 이유

$$Q = \begin{bmatrix} 1 & 2 & 1 \\ 3 & 2 & 3 \end{bmatrix}, K = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, V = \begin{bmatrix} 5 & 4 & 3 \\ 2 & 1 & 0 \end{bmatrix}$$

dot product

$$K^T V = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 0 \\ 3 & 1 & 5 \end{bmatrix} \begin{bmatrix} 5 & 4 & 3 \\ 10 & 8 & 6 \\ 15 & 12 & 9 \end{bmatrix} = \underline{\underline{5 \ 4 \ 3}}$$

$$\underbrace{f^* S_0}_1 + K^T V_1 = S_1 = \begin{bmatrix} 5 & 4 & 3 \\ 10 & 8 & 6 \\ 15 & 12 & 9 \end{bmatrix}$$

$$Q_1 \otimes S_1 = \begin{bmatrix} 5 & 4 & 3 \\ 10 & 8 & 6 \\ 15 & 12 & 9 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 & 4 & 3 \\ 20 & 16 & 12 \\ 15 & 12 & 9 \end{bmatrix} \text{ (sum)}$$

$K^T V \qquad Q$

$$= \begin{bmatrix} 40 & 32 & 24 \end{bmatrix}$$

→ 예전 방정식 풀기 흔적



Step 4.  $k_2^T V$  for  $n=2$

$$k_2^T V_2 = \begin{matrix} 4 & 8 & 4 & 0 \\ 5 & 10 & 5 & 0 \\ 6 & 12 & 6 & 0 \\ 2 & 1 & 0 \end{matrix}$$

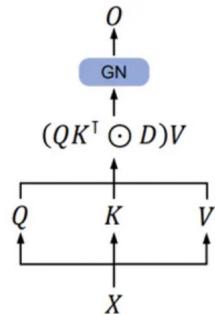
Step 5. 계수를 더해  $V$ 를 구해보자

$$f^T S_1 + k_2^T V_2 = S_2 = \begin{bmatrix} 1.25 & 1.025 \\ 2.5 & 2.15 \\ 3.115 & 3.225 \end{bmatrix} + \begin{bmatrix} 8 & 4 & 0 \\ 10 & 5 & 0 \\ -12 & 6 & 0 \end{bmatrix} = \begin{bmatrix} 9.25 & 5 & 0.25 \\ 12.5 & 7 & 1.5 \\ 15.115 & 9 & 2.25 \end{bmatrix}$$

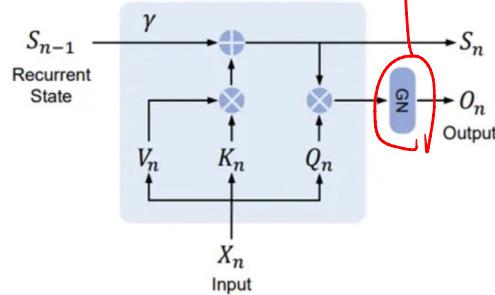
$$A_2 \otimes S_2 = \begin{bmatrix} 9.25 & 5 & 0.25 \\ 12.5 & 7 & 1.5 \\ 15.115 & 9 & 2.25 \end{bmatrix} \otimes \begin{bmatrix} 3 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 21.75 & 15 & 2.25 \\ 25 & 14 & 3 \\ 45.375 & 27 & 6.75 \end{bmatrix} = \begin{bmatrix} 100.5625 \\ 52 \\ 52 \end{bmatrix} \text{ (sum)}$$

# Final words

Batch size Normalization ~~BN~~



(a) Parallel representation.



(b) Recurrent representation.

Figure 3: Dual form of RetNet. “GN” is short for GroupNorm.

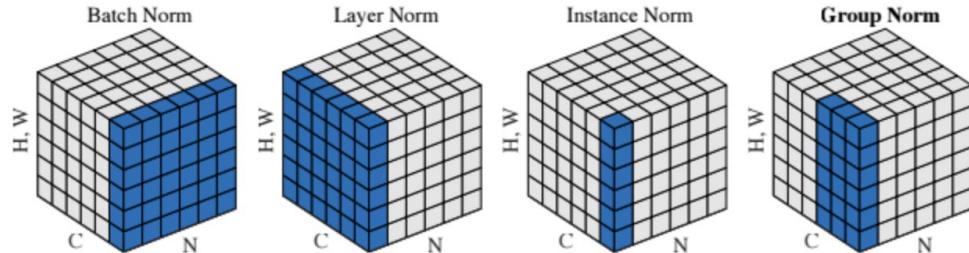


Figure 2. **Normalization methods.** Each subplot shows a feature map tensor, with  $N$  as the batch axis,  $C$  as the channel axis, and  $(H, W)$  as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

$$\hat{x}_i = \frac{1}{\sigma_i} (x_i - \mu_i)$$

$$\mu_i = \frac{1}{m} \sum_{k \in G_i} x_k.$$

$$\sigma_i = \sqrt{\frac{1}{m} \sum_{k \in G_i} (x_k - \mu_i)^2 + \epsilon}$$

$$G_i = \{k | k_N = i_N, \left[ \frac{k_C}{C/G} \right] = \left[ \frac{i_C}{C/G} \right] \}$$

$C = \text{channel}$

$G = \text{Group}$