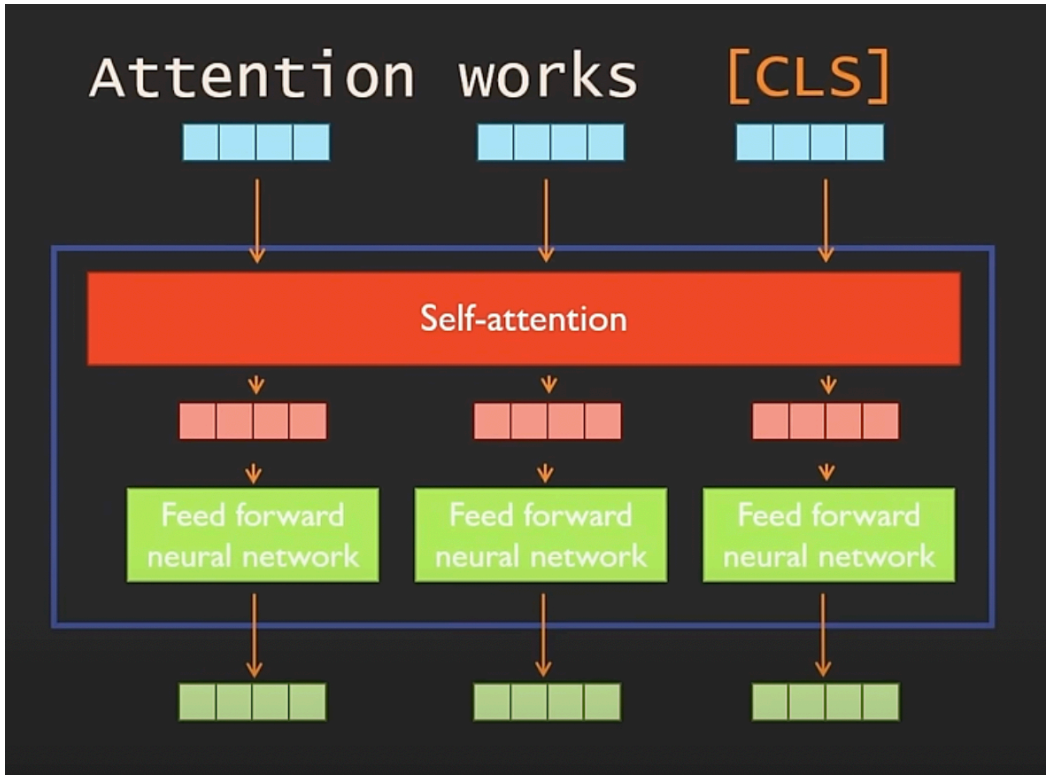
State Space Models (SSMs)

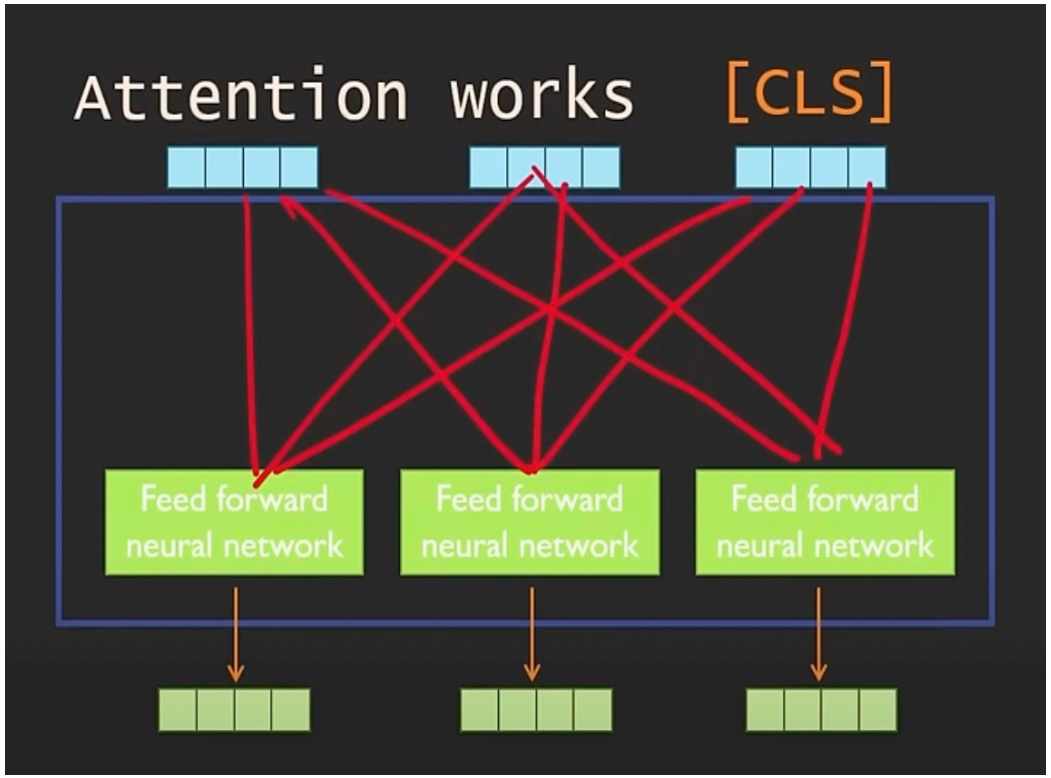
우다연
dyeonwu@gmail.com
TNT_Vision
2024/04/09

TRAIN AND TEST

Introduction

Transformer의 한계



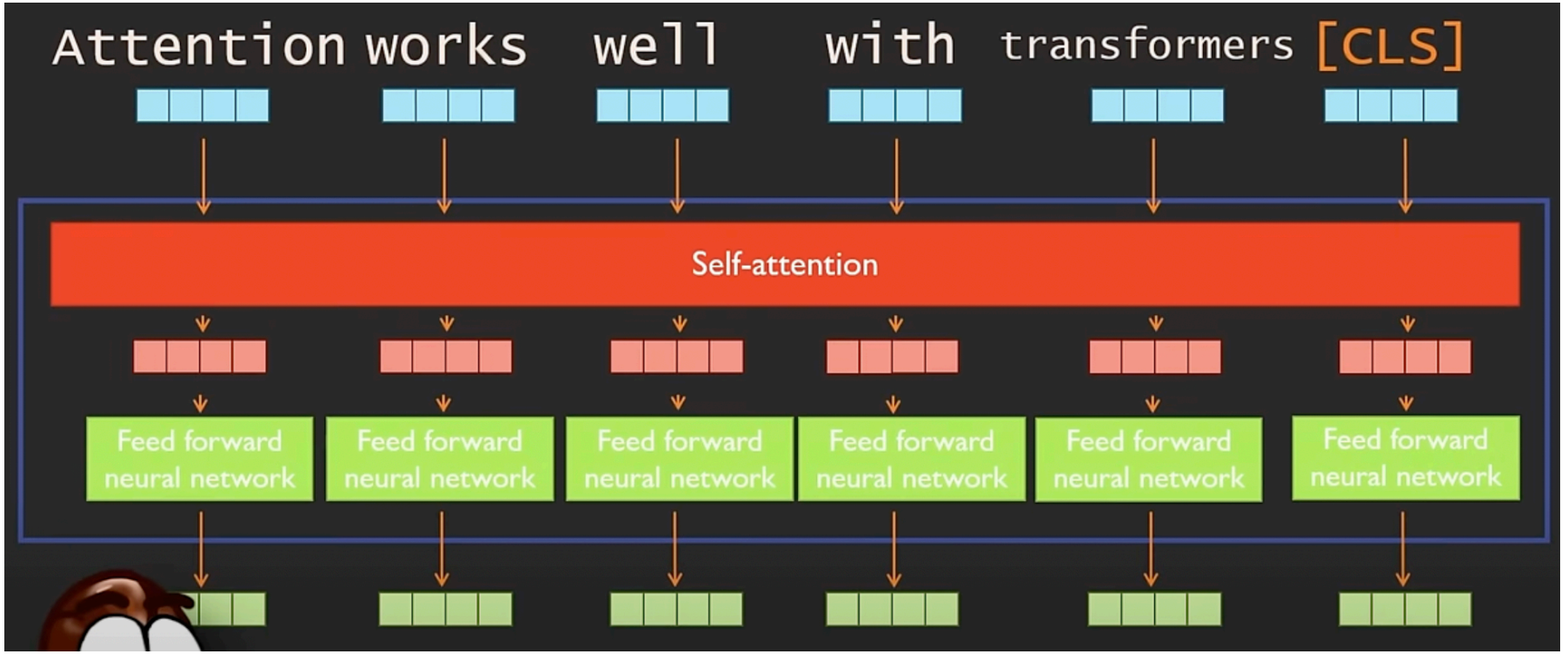


토큰 간의 관계를 일일이 구하는 Self-attention

TRAIN AND TEST

Introduction

Transformer의 한계



Sequence의 길이가 길어지면 연산량, 메모리 사용량이 Quadratic하게 증가

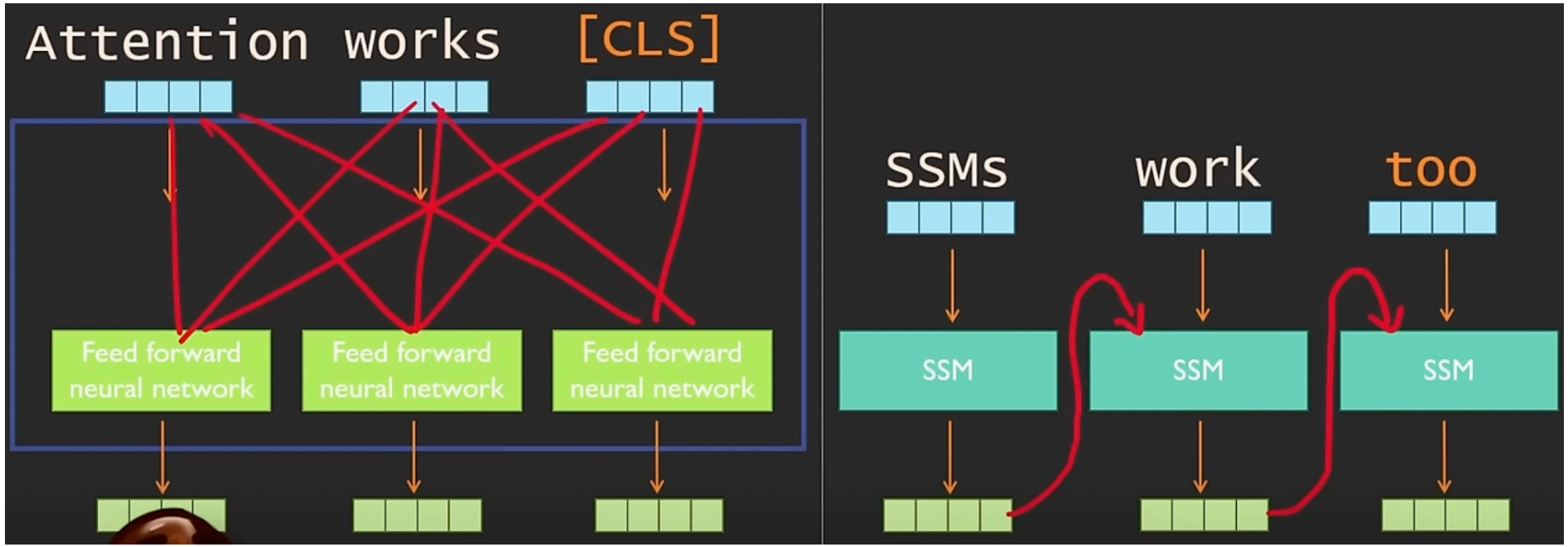
TRAIN AND TEST

Introduction

Transformer의 한계

Transformer

SSMs

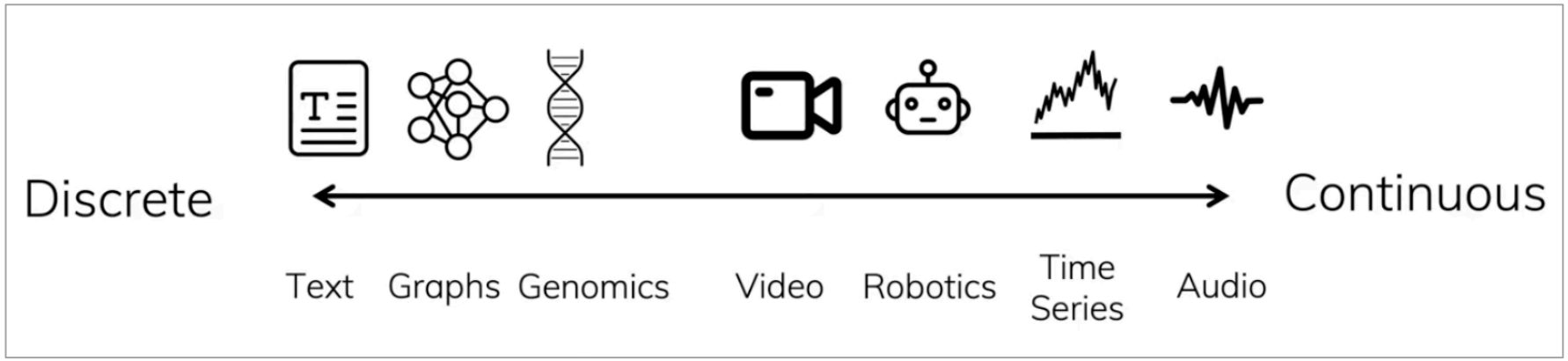


반면 SSMs는 Sequence의 길이가 길어짐에 따라 연산 비용 Linear하게 증가

TRAIN AND TEST

Introduction

Transformer의 한계



22 *unlabeled* datasets
800 GB

1 *labeled* dataset
1800 GB

데이터가 Continuous 할수록 낮은 성능을 보임

TRAIN AND TEST

Introduction

Transformer의 한계

Table 4: (**Long Range Arena**) (*Top*) Original Transformer variants in LRA. Full results in Appendix D.2. (*Bottom*) Other models reported in the literature. *Please read Appendix D.5 before citing this table.*

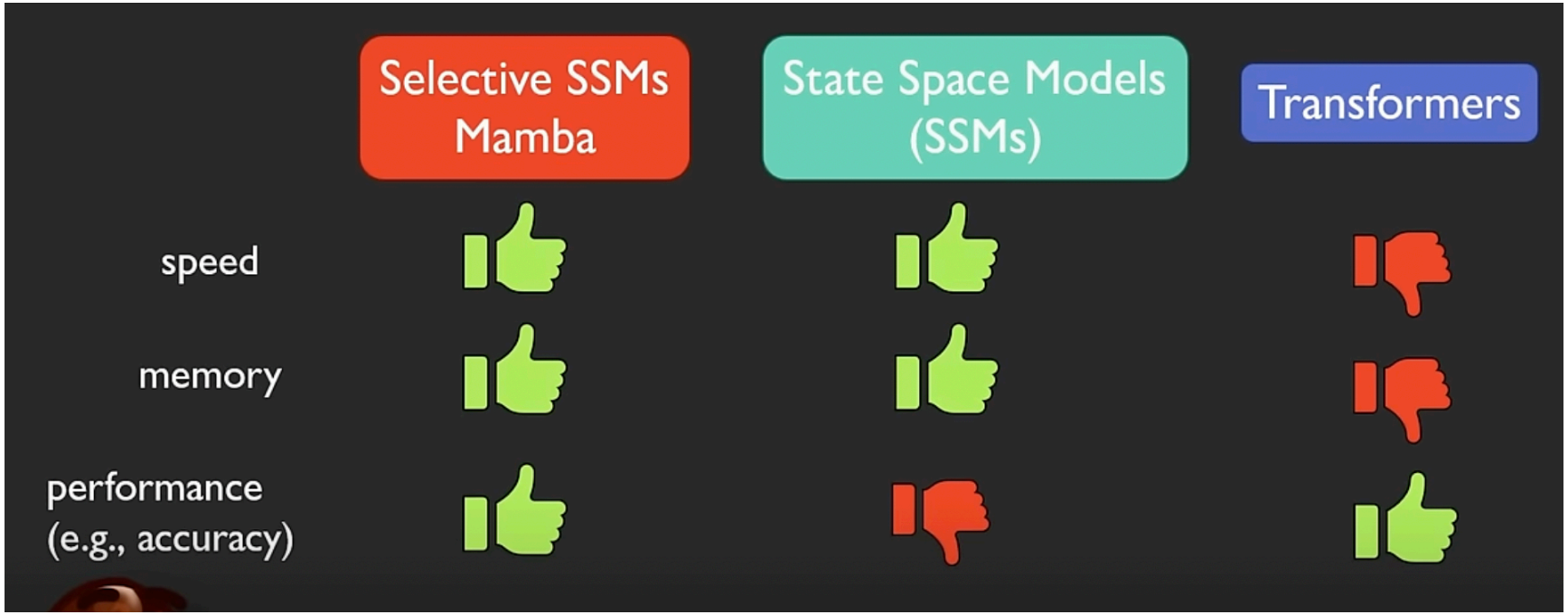
Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Transformer	36.37	64.27	57.46	42.44	71.40	✗	53.66
Reformer	37.27	56.10	53.40	38.07	68.50	✗	50.56
BigBird	36.05	64.02	59.29	40.83	74.87	✗	54.17
Linear Trans.	16.13	65.90	53.09	42.34	75.30	✗	50.46
Performer	18.01	65.40	53.82	42.77	77.05	✗	51.18
FNet	35.33	65.11	59.61	38.67	77.80	✗	54.42
Nyströmformer	37.15	65.52	79.56	41.58	70.94	✗	57.46
Luna-256	37.25	64.57	79.29	47.38	77.72	✗	59.37
S4	**59.60**	**86.82**	**90.90**	**88.65**	**94.20**	**96.35**	**86.09**

긴 Sequence에 대한 성능을 비교할 때 SSMs 기반의 S4 성능이 압도적

TRAIN AND TEST

Introduction

SSM 기반의 Mamba



Motivation: SSM 구조를 개선해서 적은 연산 비용으로 Transformer에 준하는 성능을 도출해보자!

TRAIN AND TEST

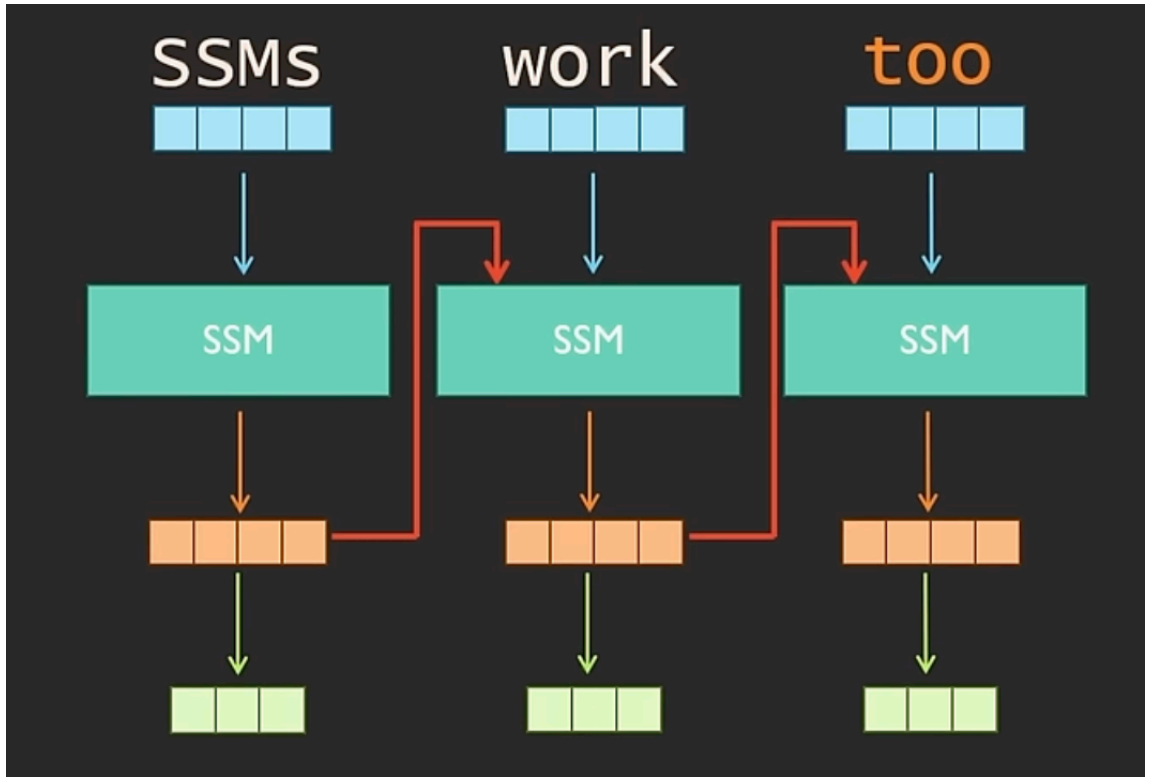
State Space Models (SSMs)

Architecture

Input $x(t)$

Hidden state $h(t)$

Output $y(t)$



$$h'(t) = Ah(t) + Bx(t) \quad (1a)$$
$$y(t) = Ch(t) \quad (1b)$$

이전 Hidden state 값 + 현재의 Input

⬇

현재 Hidden state 값 Update

⬇

Output 도출

TRAIN AND TEST

State Space Models (SSMs)

Architecture

Input $x(t) \in \mathbb{R}^1$

Hidden state $h(t) \in \mathbb{R}^n$

Output $y(t) \in \mathbb{R}^1$

$A \in \mathbb{R}^{n \times n}$

$B \in \mathbb{R}^{n \times 1}$

$C \in \mathbb{R}^{1 \times n}$

$$h'(t) = Ah(t) + Bx(t) \quad (1a)$$
$$y(t) = Ch(t) \quad (1b)$$

A: 이전 상태에서 변화할 방향과 크기를 지시하는 Linear Transformation (n -> n)

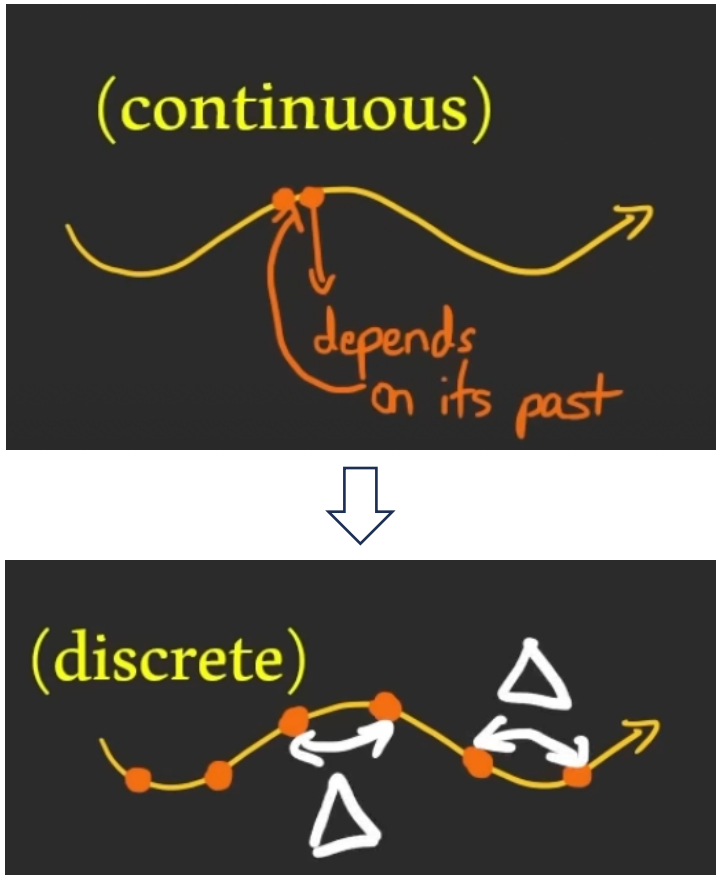
B: 새로운 입력이 어떻게 반영되어야 할지 Transform (1 -> n)

C: 모델이 내부적으로 처리한 연산을 결과물로 도출하는 Projection (n -> 1)

TRAIN AND TEST

State Space Models (SSMs)

Discretization



Δ: Step size, 토큰 간의 간격

TRAIN AND TEST

State Space Models (SSMs)

Discretization

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

$$\mathbf{x}'(t) = \lim_{\Delta \to 0} \frac{\mathbf{x}(t+\Delta) - \mathbf{x}(t)}{\Delta}$$
$$\cong \frac{\mathbf{x}(t+\Delta) - \mathbf{x}(t)}{\Delta}$$

$$\begin{aligned}\mathbf{x}(t+\Delta) &\cong \Delta(\mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)) + \mathbf{x}(t) \\ &= \Delta\mathbf{A}\mathbf{x}(t) + \Delta\mathbf{B}\mathbf{u}(t) + \mathbf{x}(t) \\ &= (\mathbf{I} + \Delta\mathbf{A})\mathbf{x}(t) + \Delta\mathbf{B}\mathbf{u}(t) \\ &= \bar{\mathbf{A}}\mathbf{x}(t) + \bar{\mathbf{B}}\mathbf{u}(t)\end{aligned}$$

$$\mathbf{x}_t = \bar{\mathbf{A}}\mathbf{x}_{t-1} + \bar{\mathbf{B}}\mathbf{u}_t$$
$$\mathbf{y}_t = \mathbf{C}\mathbf{x}_t$$

Δ는 A, B 를 변화시켜 $\bar{A}, \bar{B}$를 만듦

($\bar{A}, \bar{B}$: 이산화 되었다는 새로운 상태를 나타내기 위한 변수)

TRAIN AND TEST

State Space Models (SSMs)

Discretization

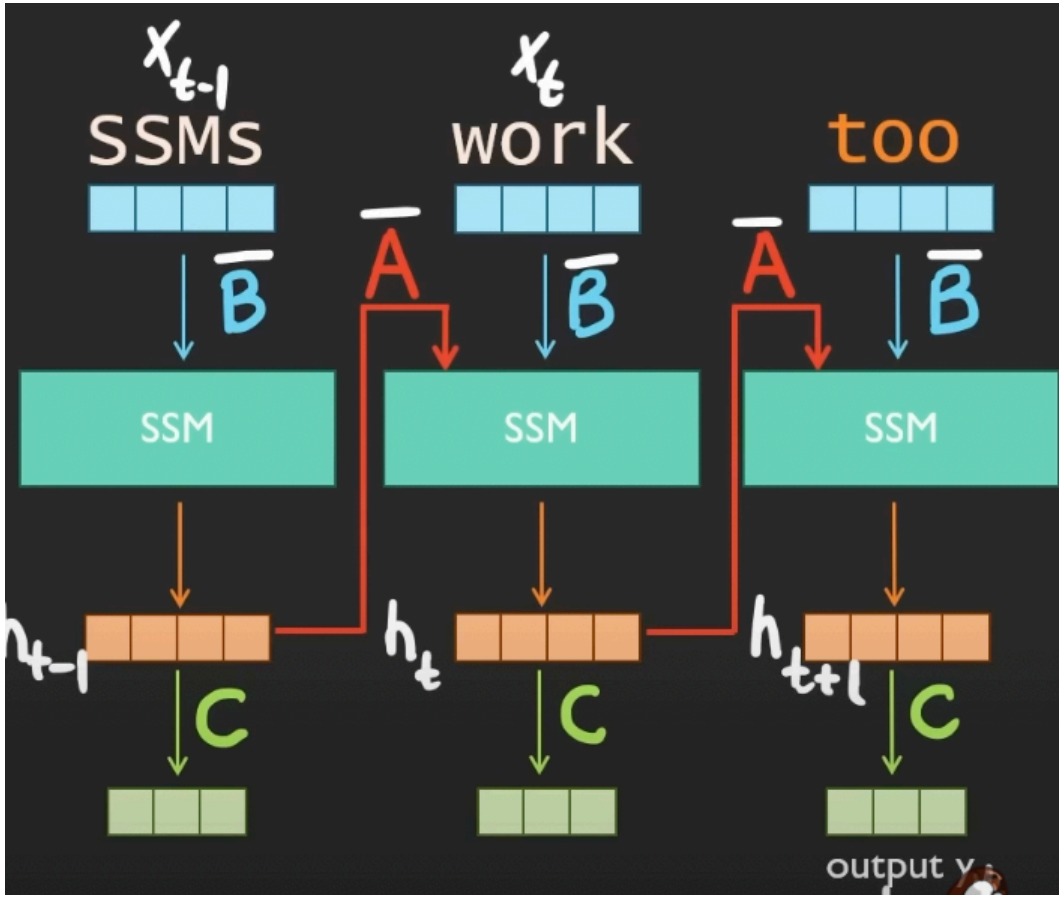
$$h_t = \overline{A}h_{t-1} + \overline{B}x_t \quad (2a)$$

$$y_t = Ch_t \quad (2b)$$

$\bar{A}$: 이전 상태에서 취해야 할 변화의 방향과 크기를 지시

$\bar{B}$: 새로운 입력이 어떻게 기여해야할지 지시

C: 모델이 내부적으로 처리한 연산을 결과물로 도출



TRAIN AND TEST

State Space Models (SSMs)

Linear Time-Invariant(LTI) system

$$h_t = \overline{A}h_{t-1} + \overline{B}x_t \qquad (2a)$$
$$y_t = Ch_t \qquad (2b)$$

$$\overline{A} = \exp(\Delta A) \qquad \overline{B} = (\Delta A)^{-1}(\exp(\Delta A) - I) \cdot \Delta B$$

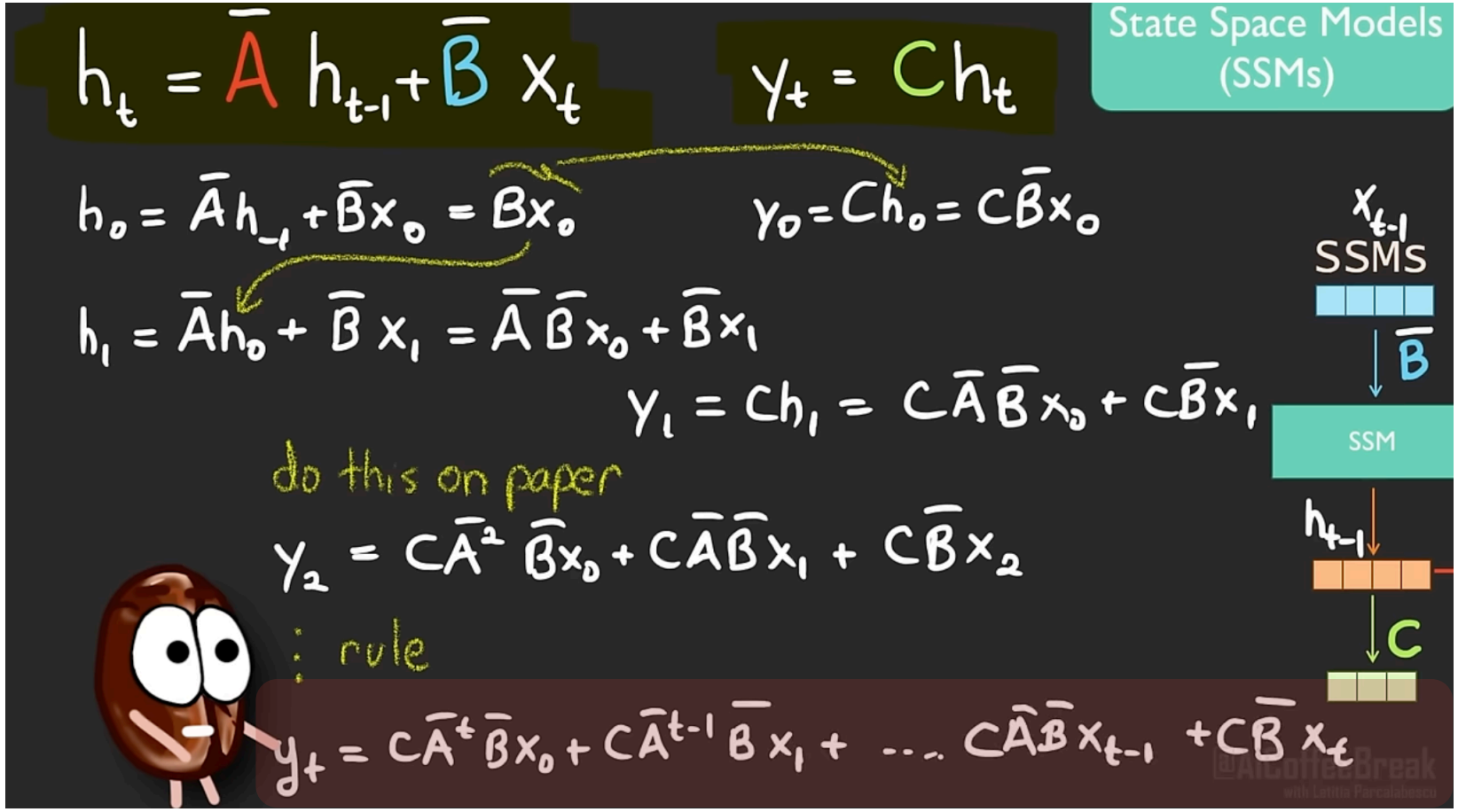
Zero-order Hold (ZOH) 방식으로 정의한 $\bar{A}, \bar{B}$

$\bar{A}, \bar{B}, C$: 모두 고정적인 값

TRAIN AND TEST

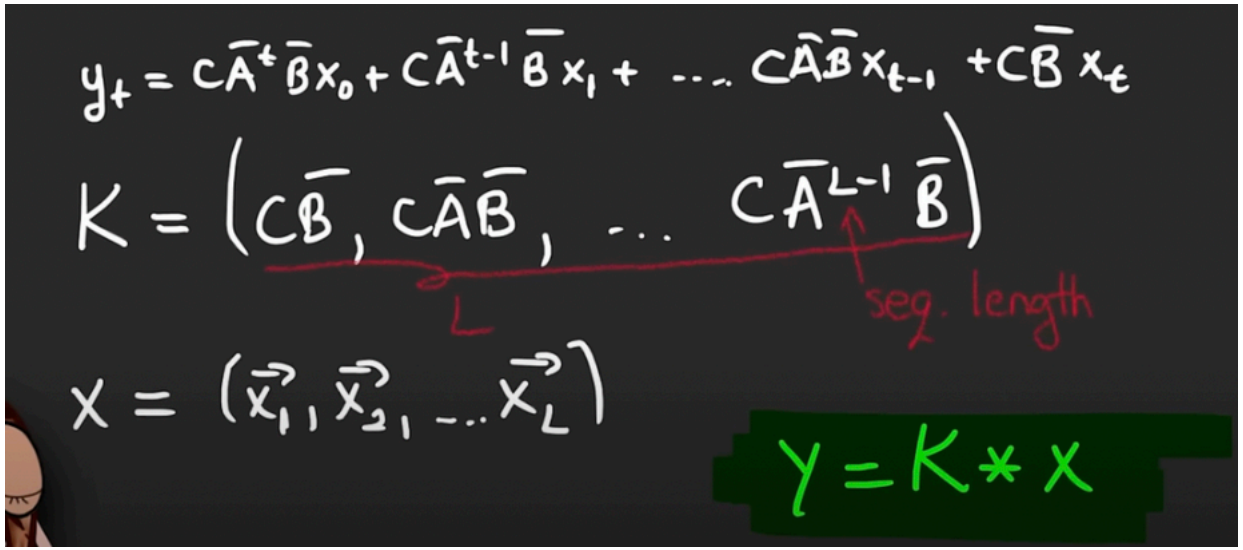
State Space Models (SSMs)

LTI system



State Space Models (SSMs)

LTI system



y: 일반화된 Output 도출 공식

K(Kernel): 미리 알고 있는 변수들의 조합

X(Input): Input 토큰의 조합

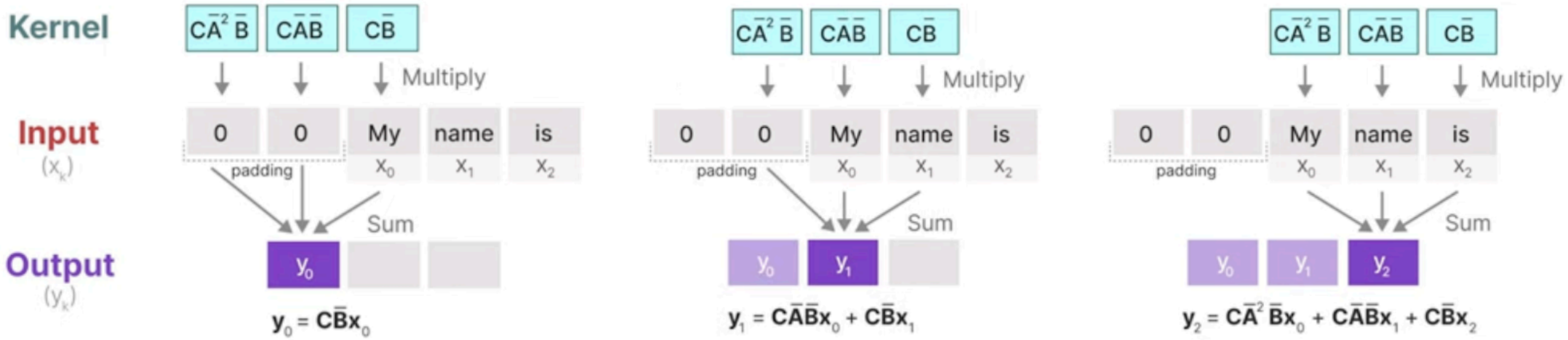
⇨ $y = K * X$

값이 고정되어 있고 식을 미리 구해둘 수 있기 때문에 RNN에 비해 빠른 연산 가능

TRAIN AND TEST

State Space Models (SSMs)

LTI system



$\bar{A}, \bar{B}, C$ 의 조합인 Kernel이 필터처럼 곱해진다는 맥락에서

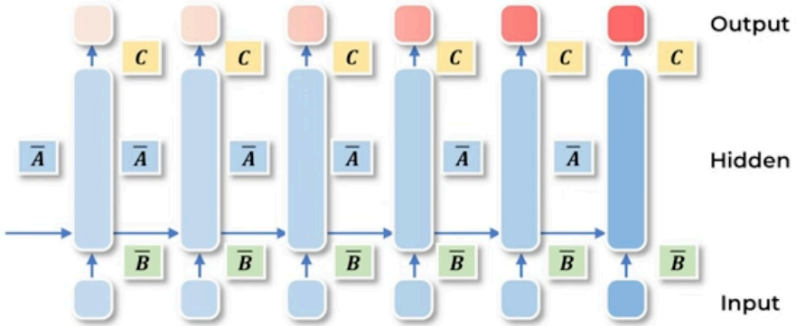
Recurrence를 Convolutionize 하는 형태로도 볼 수 있음

TRAIN AND TEST

State Space Models (SSMs)

LTI system

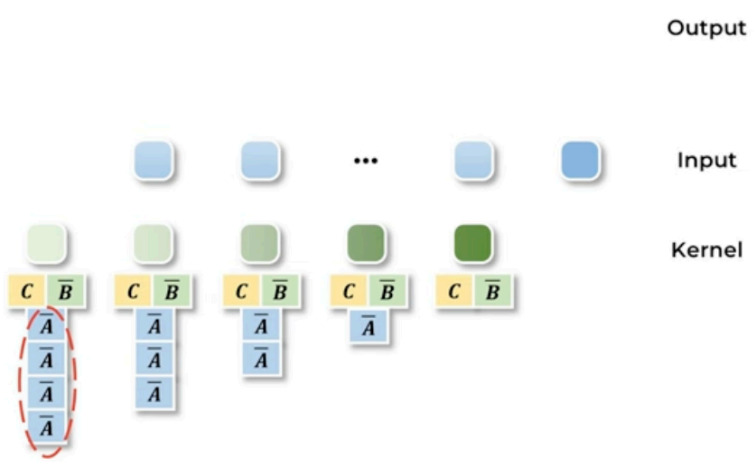
[Recurrent]



$$\mathbf{A} \in \mathbb{R}^{n \times n}$$
$$\mathbf{B} \in \mathbb{R}^{n \times 1}$$
$$\mathbf{C} \in \mathbb{R}^{1 \times n}$$

한 번에 한 time-stamp에 접근해야 할 때

[Convolution]

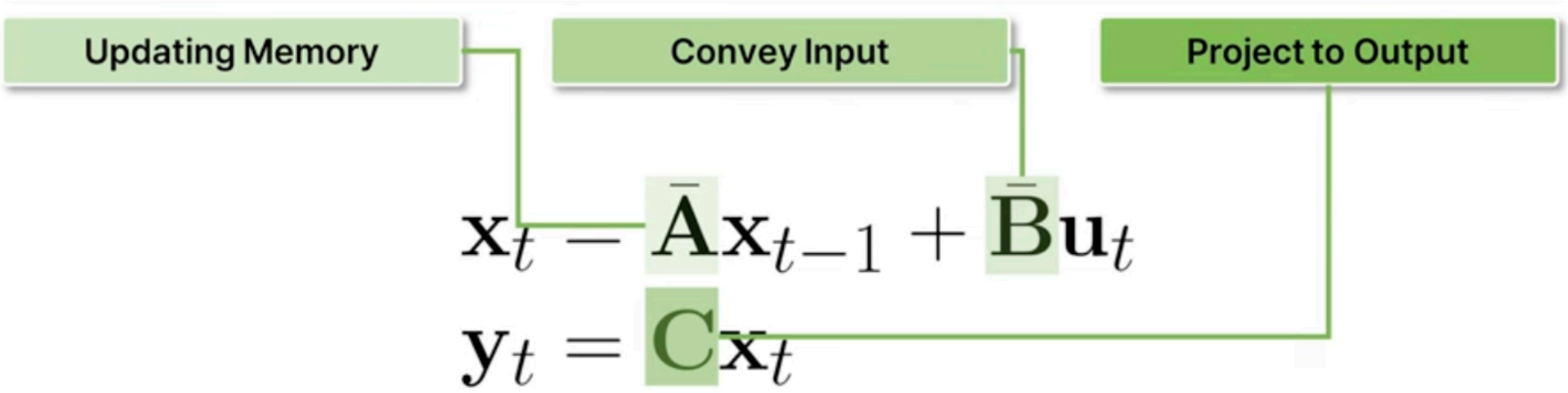


전체 Input을 미리 파악할 수 있을 때

TRAIN AND TEST

State Space Models (SSMs)

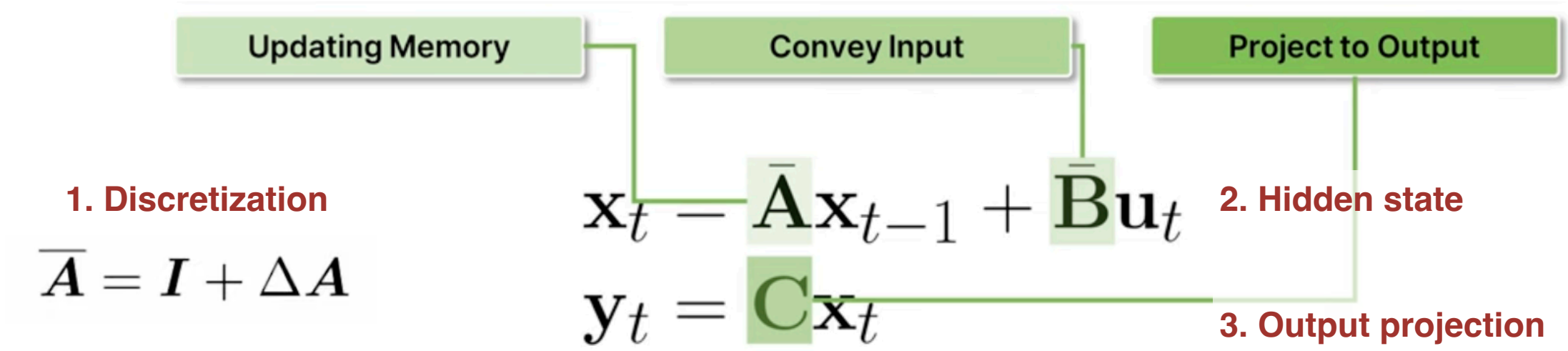
Review



TRAIN AND TEST

State Space Models (SSMs)

Review

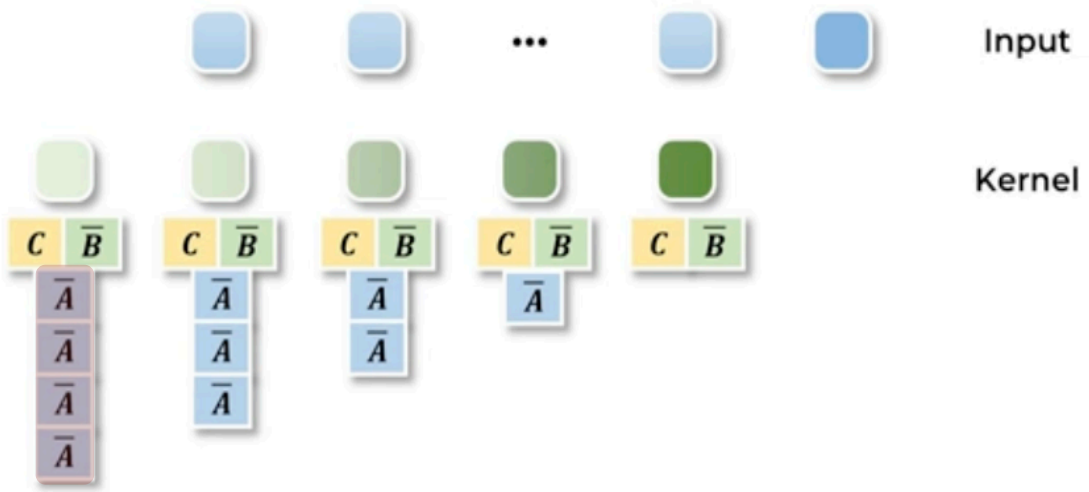


TRAIN AND TEST

Structured State Space Model (S4)

Motivation

$$h'(t) = Ah(t) + Bx(t) \quad (1a)$$
$$y(t) = Ch(t) \quad (1b)$$



N * N 형태의 A의 값을 어떻게 정의하면 좋을지?

기존 SSMs에서 연산 효율을 높일 수 있을지?

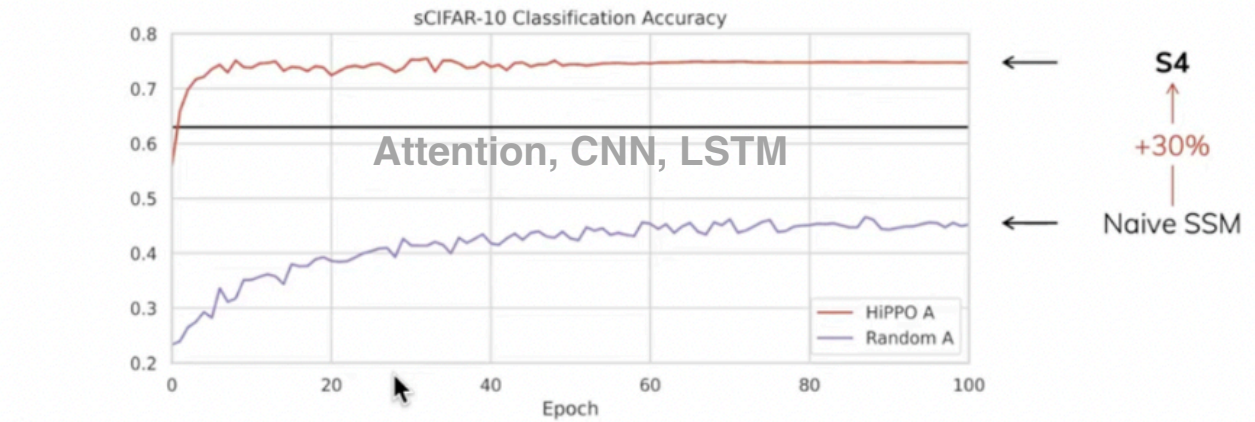
TRAIN AND TEST

Structured State Space Model (S4)

Motivation

$$h'(t) = Ah(t) + Bx(t) \quad (1a)$$
$$y(t) = Ch(t) \quad (1b)$$

N * N 형태의 A의 값을 어떻게 정의하면 좋을지?



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 3 & 3 & 0 \\ 1 & 3 & 5 & 4 \end{bmatrix}$$

$$x' = Ax + Bu$$

예) HiPPO: A, B에 대해 고정된 값의 행렬을 활용했음

학습 없이도 다른 모델보다 높은 성능을 보임

TRAIN AND TEST

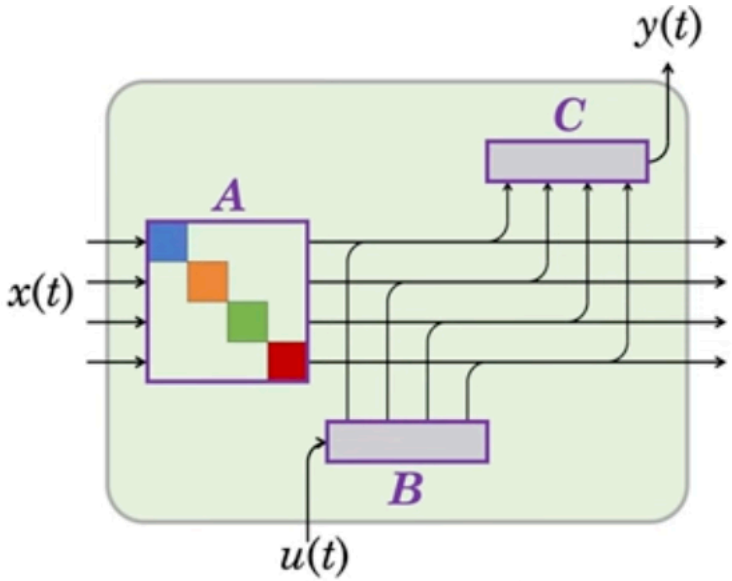
Structured State Space Model (S4)

Structure

A 대각행렬화 & Element-wise multiplication

⇨ **연산 효율 높임**

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix} \qquad x = \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}$$



S4D Recurrent View

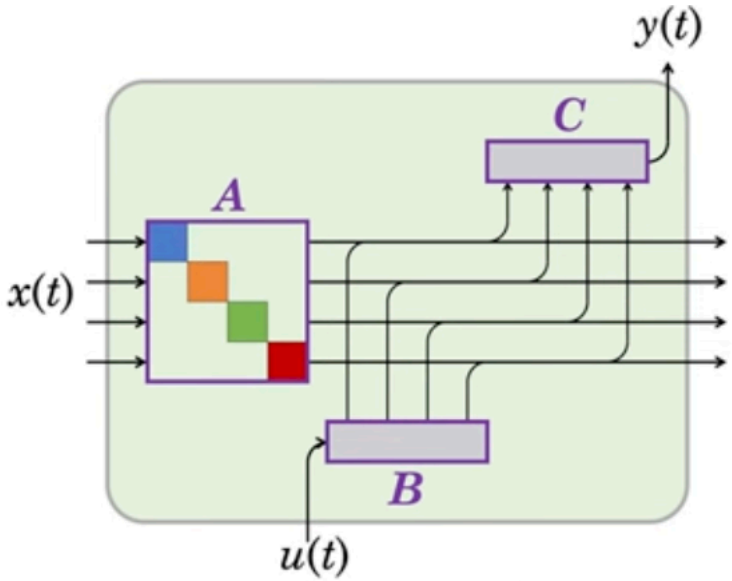
Structured State Space Model (S4)

Limitation

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix} \qquad x = \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}$$

데이터의 채널, 즉 행렬의 차원이 늘어나면

모델이 관리해야 하는 독립적인 값이 많아짐 => 병목 발생



S4D Recurrent View

TRAIN AND TEST

참고

https://www.youtube.com/watch?v=vrF3MtGwD0Y

https://www.youtube.com/watch?v=luCBXCErkCs

https://www.youtube.com/watch?v=JjxBNBzDbNk

TRAIN AND TEST

TRAIN AND TEST

Mamba:

Linear-Time Sequence Modeling with Selective State Spaces

TNT 24-1 CV
우다연, 서강민, 조재희
24.04.09

TRAIN AND TEST

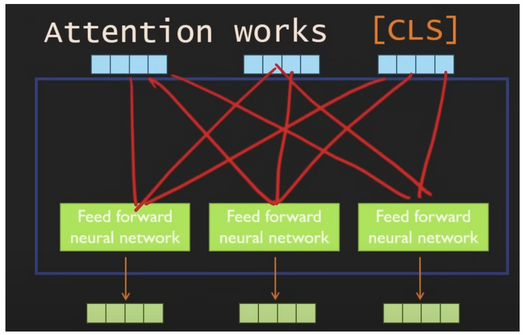
SSM + Selection(S6)

Motivation : Selection as a Means of Compression

Transformer

context 압축 X, quadratic한 attention 연산 / 대신 성능 good

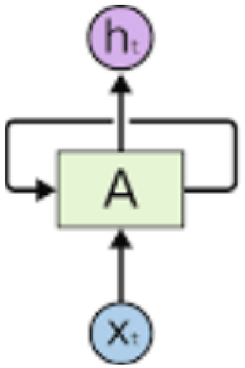
⇒ 성능을 유지하면서 속도를 높이려는 연구



Recurrent Models (RNN, SSM, ...)

한정된 context, linear한 recurrent 연산 / 대신 성능 제한됨

⇒ 속도를 유지하면서 성능을 높이려는 연구



State가 한정된 Recurrent model에서는 State를 잘 관리해야 한다!

TRAIN AND TEST

SSM + Selection(S6)

Motivation : Selection as a Means of Compression

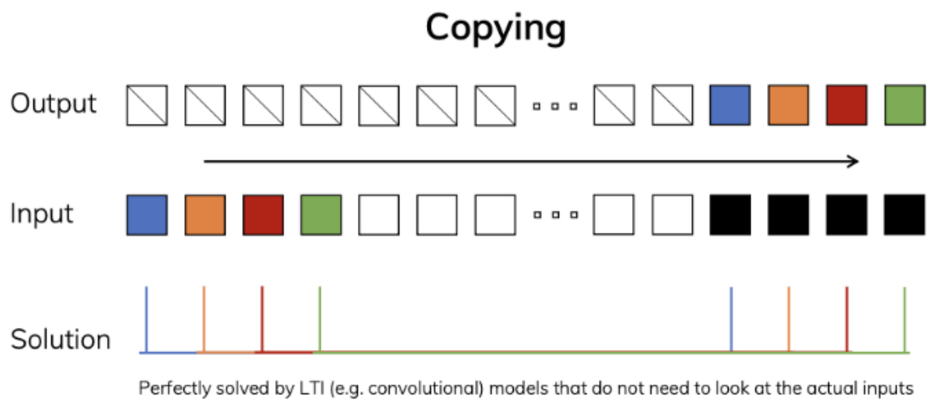
LTI 모델의 한계점을 드러내는 Task

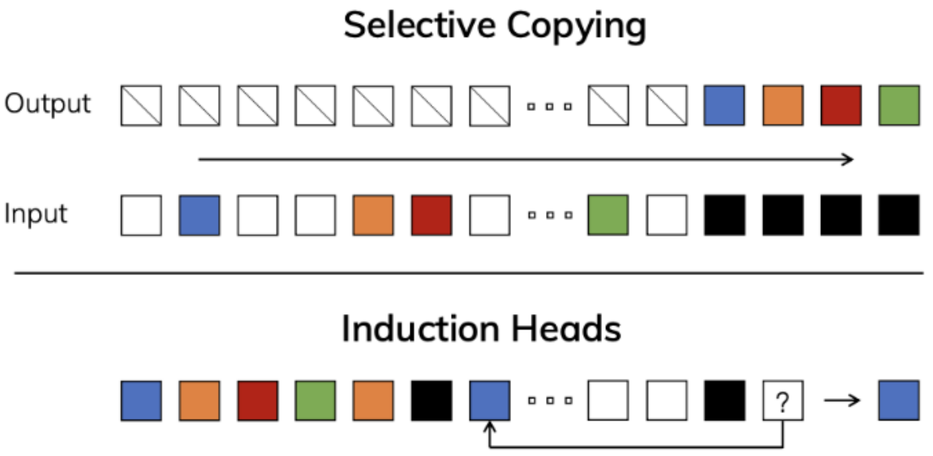
Selective Copying : 기억해야 할 토큰들 위치가 매번 바뀜

- 흰색 블럭과 색깔 블럭을 구분할 수 있는 능력이 필요(content-awareness)

Induction Heads : 검정색 다음에 파란색이 온다는 정보를 기억해야 함

- context(현재 검정색)에 따라 다음 답변(파란색)을 만드는 능력(context-awareness)





TRAIN AND TEST

SSM + Selection(S6)

Motivation : Selection as a Means of Compression

Input과 context에 따라 적절한 행동을 하는 능력이 부족함
→ Selectivity를 부여하자!

Selective
Structured
State
Space
Sequence Model
+ **S**can

=S6

SSSSSS....

TRAIN AND TEST

SSM + Selection(S6)

Improving SSMs with Selection

S6 알고리즘 주요 차이점

Algorithm 1 SSM (S4)

Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
1: $\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
3: $\boldsymbol{C} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
4: $\Delta : (\mathtt{D}) \leftarrow \tau_\Delta$(Parameter)
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{D}, \mathtt{N}) \leftarrow$ discretize$(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \mathrm{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
7: **return** y

Algorithm 2 SSM + Selection (S6)

Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
1: $\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_B(x)$
3: $\boldsymbol{C} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_C(x)$
4: $\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta$(Parameter$+s_\Delta(x)$)
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N}) \leftarrow$ discretize$(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \mathrm{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
7: **return** y

- **LTI Model → Time-Variant Model로 변환**

  Δ, B, C 차원에 L 추가

  이제 일관된 값이 아니라 Time Step마다 다른 값 사용

TRAIN AND TEST

SSM + Selection(S6)

Improving SSMs with Selection

S6 알고리즘 주요 차이점

Algorithm 1 SSM (S4)

Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
1: $\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
3: $\boldsymbol{C} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
4: $\Delta : (\mathtt{D}) \leftarrow \tau_\Delta$(Parameter)
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{D}, \mathtt{N}) \leftarrow$ discretize$(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \mathrm{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
7: **return** y

Algorithm 2 SSM + Selection (S6)

Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
1: $\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_B(x)$
3: $\boldsymbol{C} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_C(x)$
4: $\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta$(Parameter$+s_\Delta(x)$)
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N}) \leftarrow$ discretize$(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \mathrm{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
7: **return** y

- Δ, B, C - **일반 Paramter → input에 대한 함수로 변환**

Δ - $\gamma_\Delta(Parameter + s_\Delta(x))$,

$s_\Delta(x) = Broadcast_D(Linear_1(x))$

B, C - $S_B(x) = Linear_N(x), S_C(x) = Linear_N(x)$

TRAIN AND TEST

SSM + Selection(S6)

Selective Parameters

- Δ - $\gamma_\Delta(Parameter + s_\Delta(x))$, $s_\Delta(x) = Broadcast_D(Linear_1(x))$
  - $s_\Delta(x)$ - input x를 scalar로 만든 뒤 D차원(embedding 차원)으로 broadcast
  - 현재 input에 얼마나 집중하거나 무시할 지 밸런스를 결정
  - Δ가 크면 state h를 reset하고 현재 input x에 집중함, 작으면 반대
    - $h(t+\Delta)$
      $= h(t)(I + \Delta A) + \Delta B x(t)$
      $= \overline{A}h(t) + \overline{B}x(t)$
      여기서 Δ가 0이면 $x(t)$ 지분이 0이 되고, Δ가 클수록 $x(t)$ 지분이 커짐
  - 일반적인 RNN gating mechanism 역할을 한다고 볼 수 있음

SSM + Selection(S6)

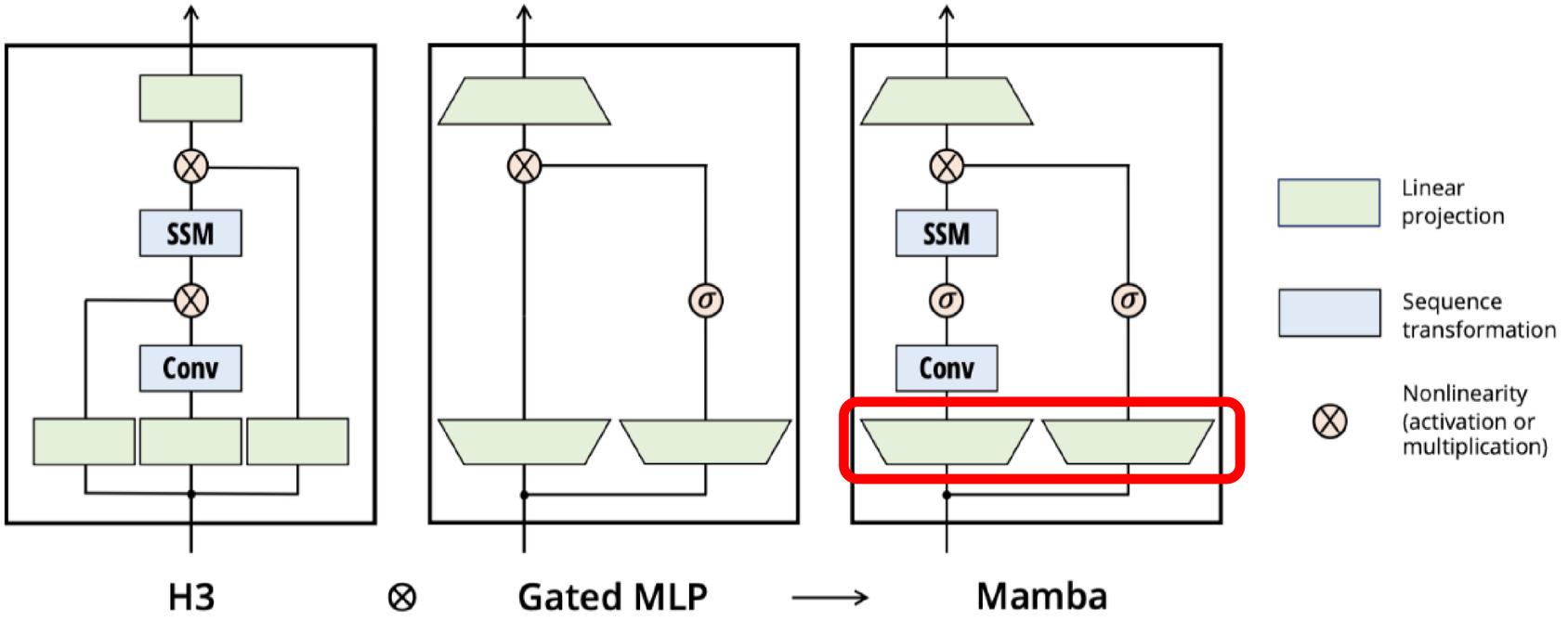
Selective Parameters

- $A - Parameter$
  - 이전 state를 다음 state에 반영하는 역할
  - A에도 selectivity를 부여할 수 있지만, 이미 Δ를 거치므로 simplicitiy를 위해 굳이 하지 않음

- B, C - $S_B(x) = Linear_N(x), S_C(x) = Linear_N(x)$
  - B - input을 transform, C - output을 transfrom
  - B와 C에 selectivity를 부여함으로써 더 미세한 조정(finer-grained control)을 가능케 함

SSM + Selection(S6)

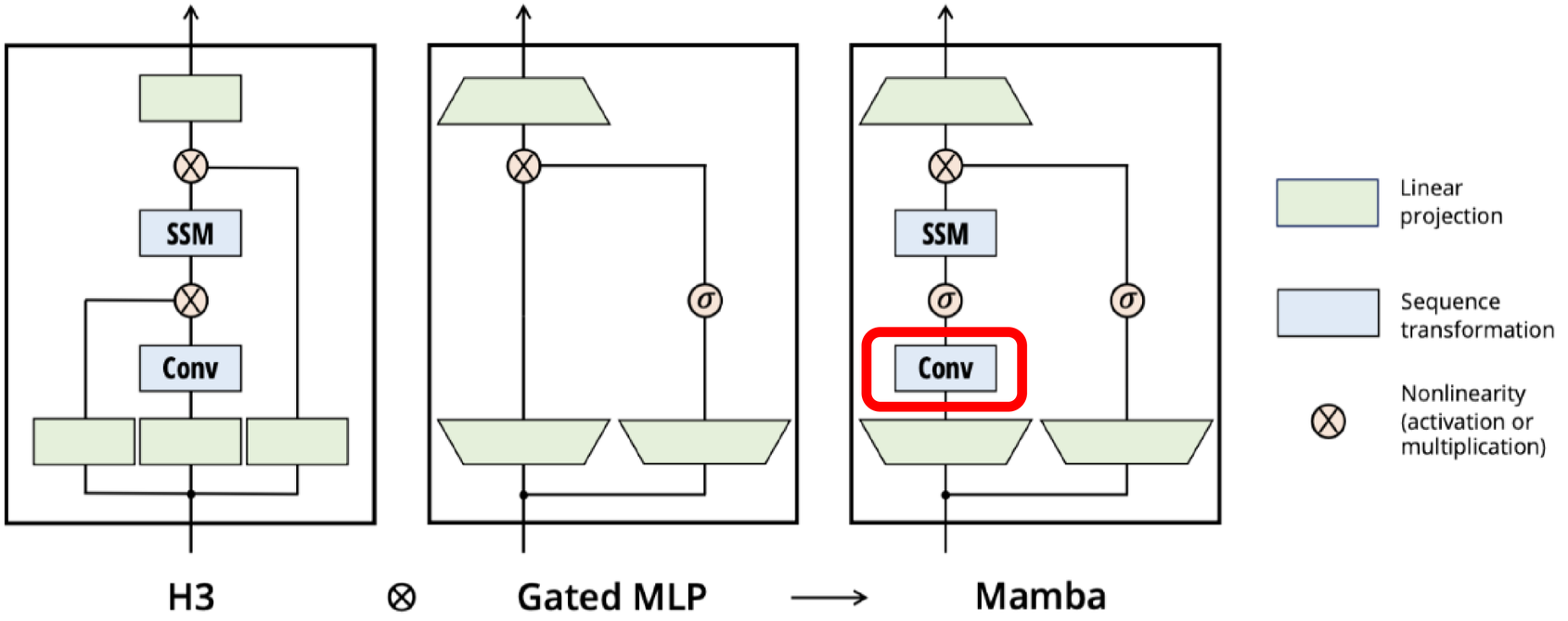
A Simplified SSM Architecture



- Linear projection : input 차원을 2배로 확장 (표현력 증가)

SSM + Selection(S6)

A Simplified SSM Architecture

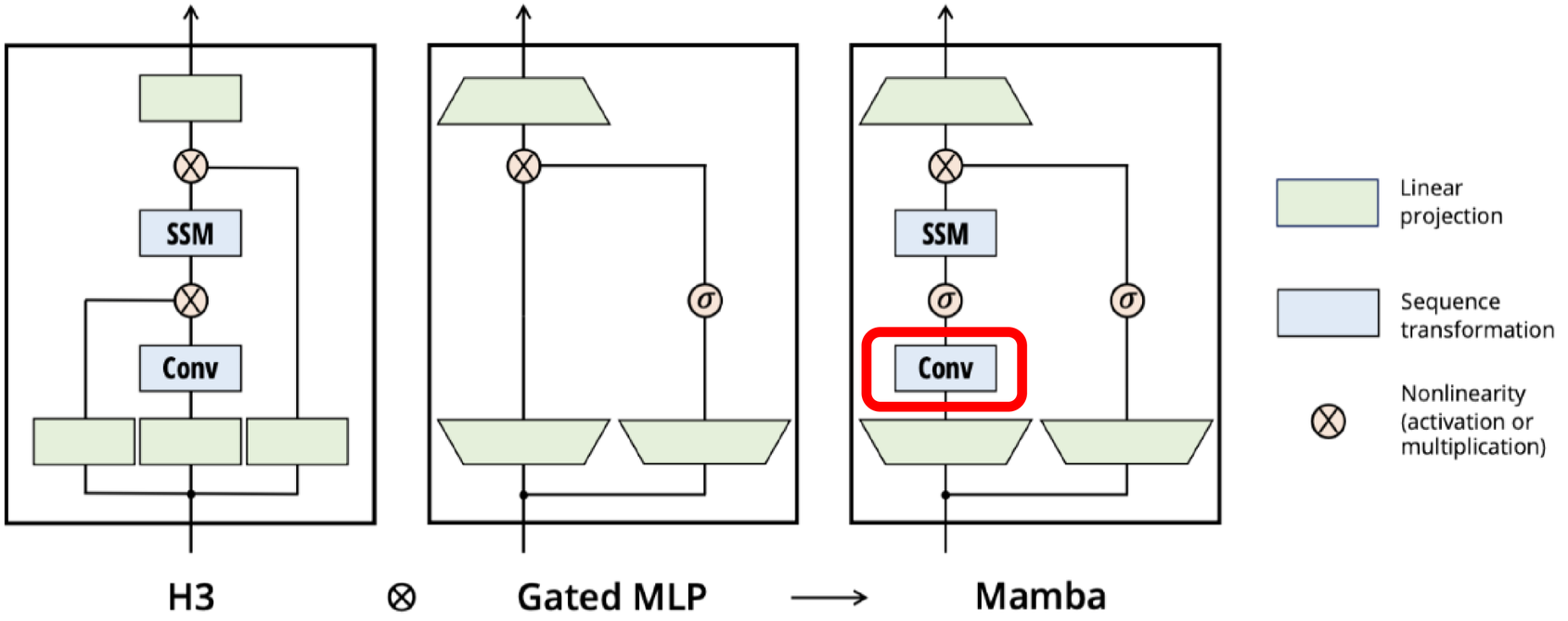


- Canonical 1d Conv : 차원 간에 정보를 고르게 확산시키는 역할

TRAIN AND TEST

SSM + Selection(S6)

A Simplified SSM Architecture

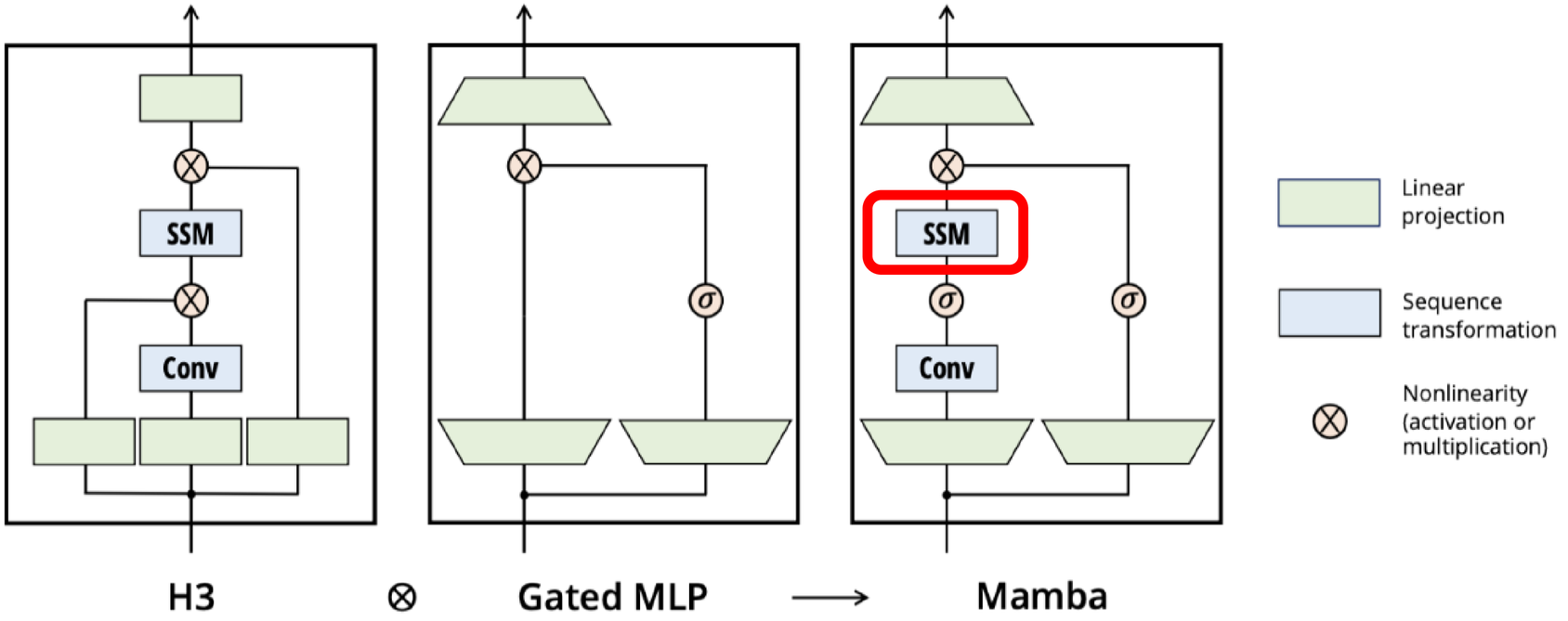


- Canonical 1d Conv : 차원 간에 정보를 고르게 확산시키는 역할

TRAIN AND TEST

SSM + Selection(S6)

A Simplified SSM Architecture

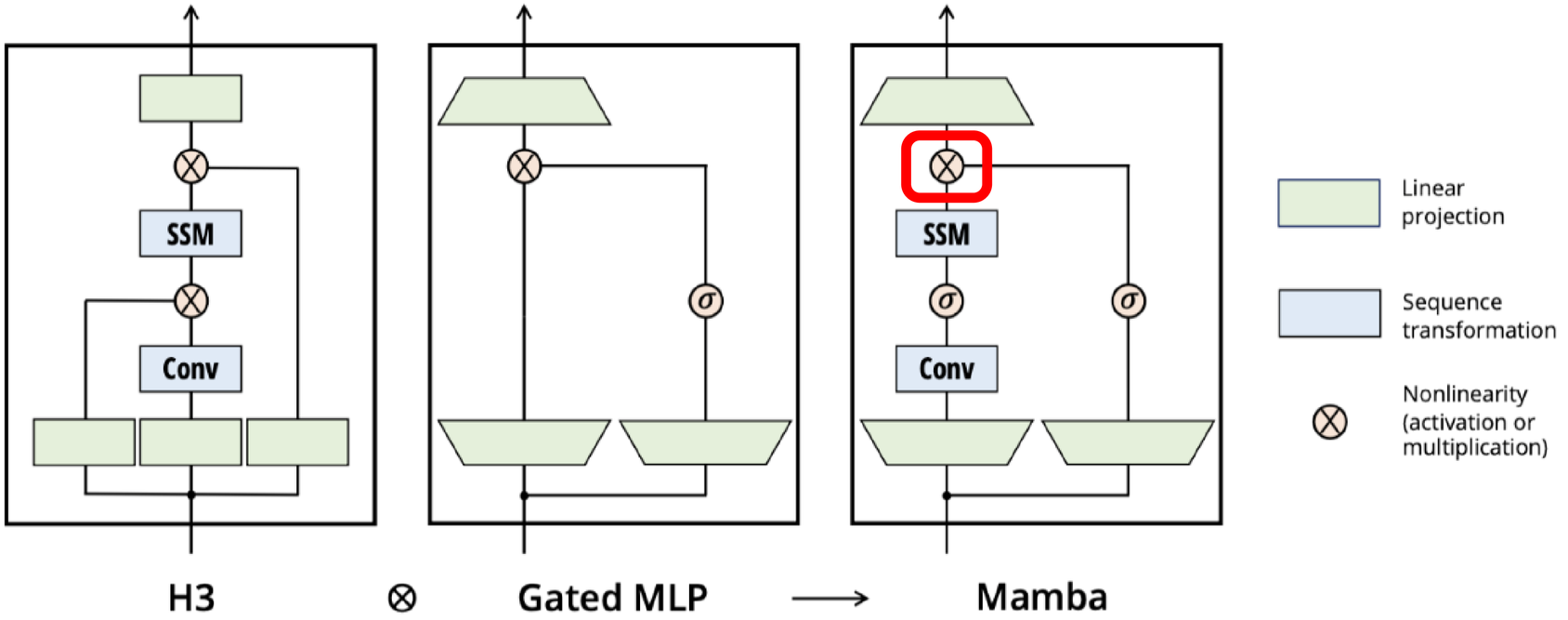


- S6 block

TRAIN AND TEST

SSM + Selection(S6)

A Simplified SSM Architecture

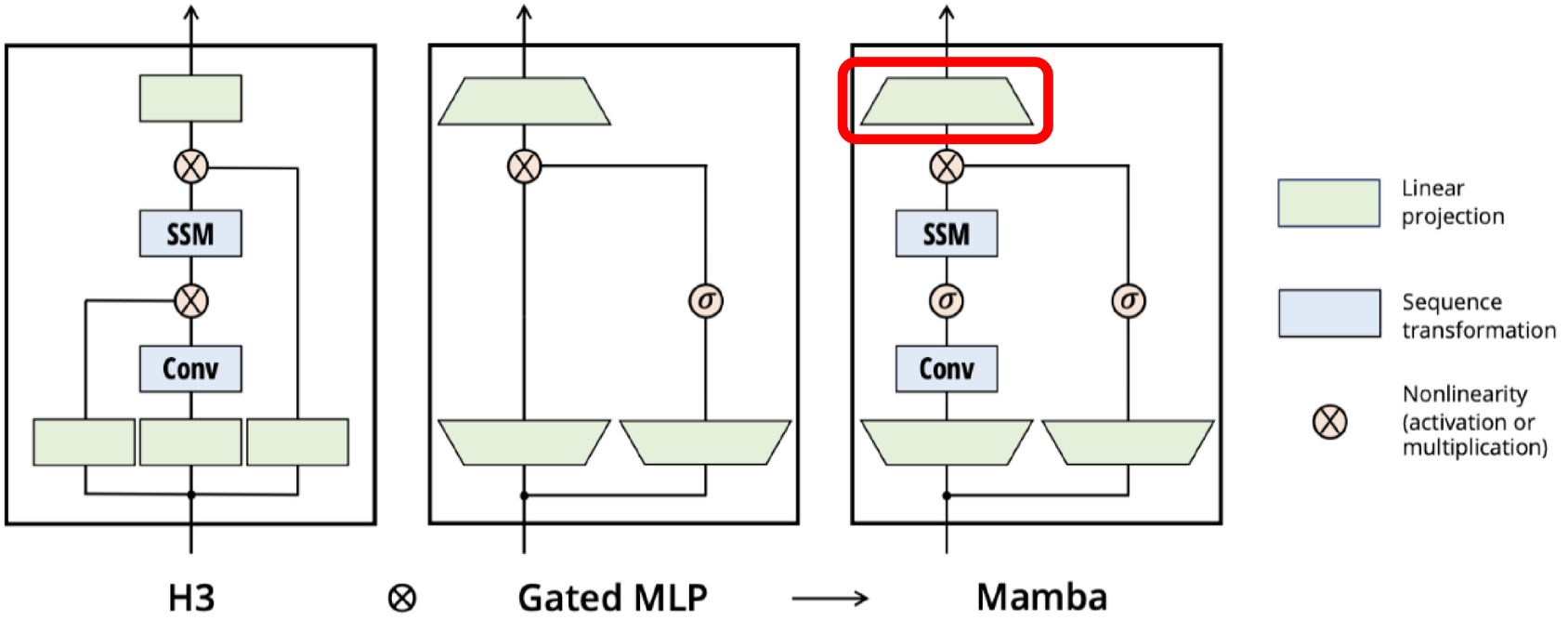


- Multiplication :
  이전 token들의 정보를 담고 있는 SSM output(왼쪽)과
  현재 token의 embedding(오른쪽) 간의 유사도 계산하는 역할

TRAIN AND TEST

SSM + Selection(S6)

A Simplified SSM Architecture



- Linear projection : output 차원을 다시 원래 input 차원으로 축소(1/2)

SSM + Selection(S6)

Properties of Selection Mechanism

Variable Spacing

- “um”과 같은 불필요한 input 걸러내기

SSM + Selection(S6)

Properties of Selection Mechanism

Boundary Resetting

- ex) 독립된 두 개의 문장을 한번에 번역할 때, 문장들의 정보가 섞이지 않도록 해야 함

- Transformer - attention mask로 구현
- RNN - 불가능
- Mamba - Selectivity를 통해 boundary에서 state를 reset할 수 있음

SSM + Selection(S6)

Efficient Implementation of Selective SSMs

- Selectivity 도입 이후 SSM Parameter들이 매 time step마다 바뀜

$$CAAAAAAB \rightarrow CA_0A_1A_2A_3A_4A_5A_6B,$$

→ Convolution 불가능

(Kernel을 미리 만들 수 없음)

$$\overline{\boldsymbol{K}} = (C\overline{\boldsymbol{B}}, C\overline{\boldsymbol{AB}}, \dots, C\overline{\boldsymbol{A}}^{k}\overline{\boldsymbol{B}}, \dots)$$

TRAIN AND TEST

SSM + Selection(S6)

Efficient Implementation of Selective SSMs

- Selective SSM의 계산 효율을 위해 3가지의 테크닉을 도입

1. Kernel Fusion
2. Parallel Associative Scan
3. Recomputation

SSM + Selection(S6)

Efficient Implementation of Selective SSMs

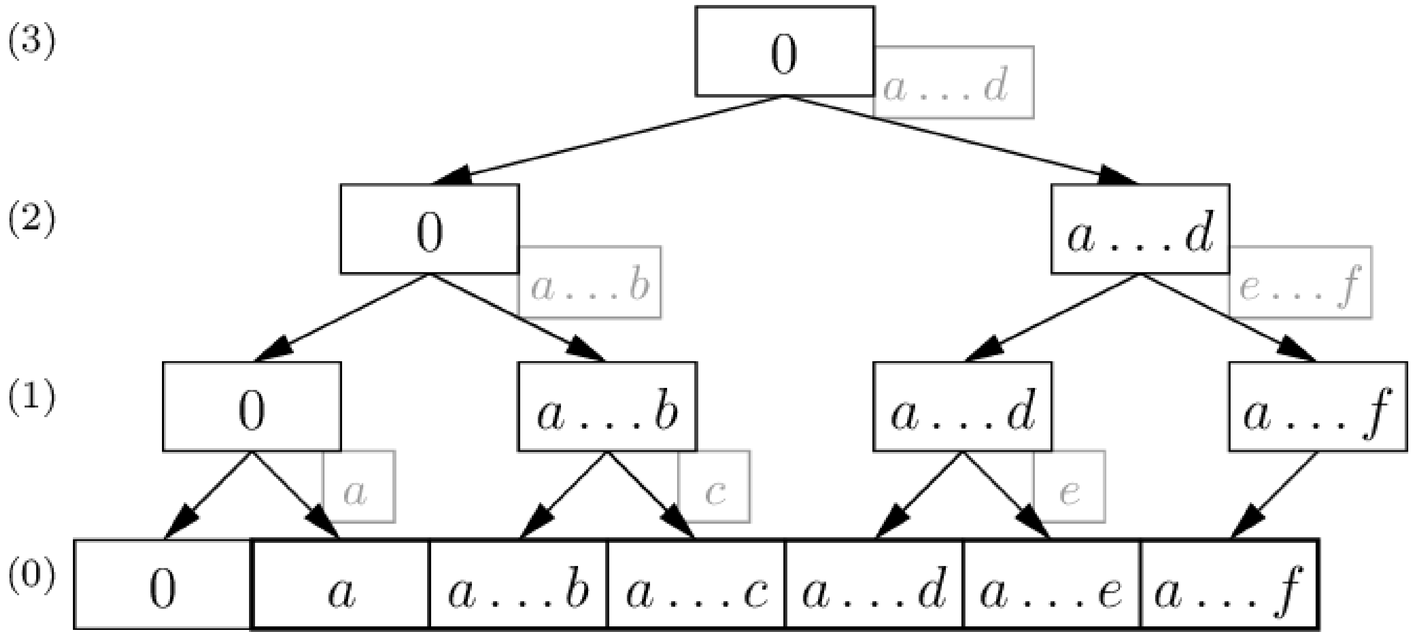
Parallel Associative Scan

- Parallel : 병렬로 $CA_0A_1A_2A_3A_4A_5A_6B\cdots$ 를 계산함
  (Blelloch 알고리즘)
- Associative : 결합법칙이 성립하는
  (덧셈, 곱셈 등)
- Scan : 누적 연산
  (다이나믹 프로그래밍과 비슷)

SSM + Selection(S6)

Efficient Implementation of Selective SSMs

Blelloch Algorithm



TRAIN AND TEST


TNT
TRAIN AND TEST

Mamba: Linear – Time Sequence Modeling With Selective State space

조재희

2024/04/09

TRAIN AND TEST

*Overview of selective Scan: Hardware-Aware State Expansion

- SSM의 고질적인 computation problem을 해결하려고 함
- sol 1) kernel fusion
- sol 2) parallel scan
- sol3) recomputation

- 문제 1: sequential nature of recurrence
- 문제 2: large memory usage
- ⇒ state h를 full로 사용하지는 않을 것

TRAIN AND TEST

2. Kernel Fusion

- Kernel funsion

GPU관점과 memory hierachy 관점에서 더 효율적인 레벨 때에 state “h”를 이용

: kernel fusion 을 사용하여 memory IOs의 양을 줄일 것 → 속도 향상

Algorithm 2 SSM + Selection (S6)

Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
1: $\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_B(x)$
3: $\boldsymbol{C} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_C(x)$
4: $\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
7: **return** y

$$h_t = \overline{\boldsymbol{A}} h_{t-1} + \overline{\boldsymbol{B}} x_t \quad (2a)$$
$$y_t = \boldsymbol{C} h_t \quad (2b)$$

$$\overline{\boldsymbol{A}} = \exp(\Delta \boldsymbol{A}) \qquad \overline{\boldsymbol{B}} = (\Delta \boldsymbol{A})^{-1}(\exp(\Delta \boldsymbol{A}) - \boldsymbol{I}) \cdot \Delta \boldsymbol{B}$$

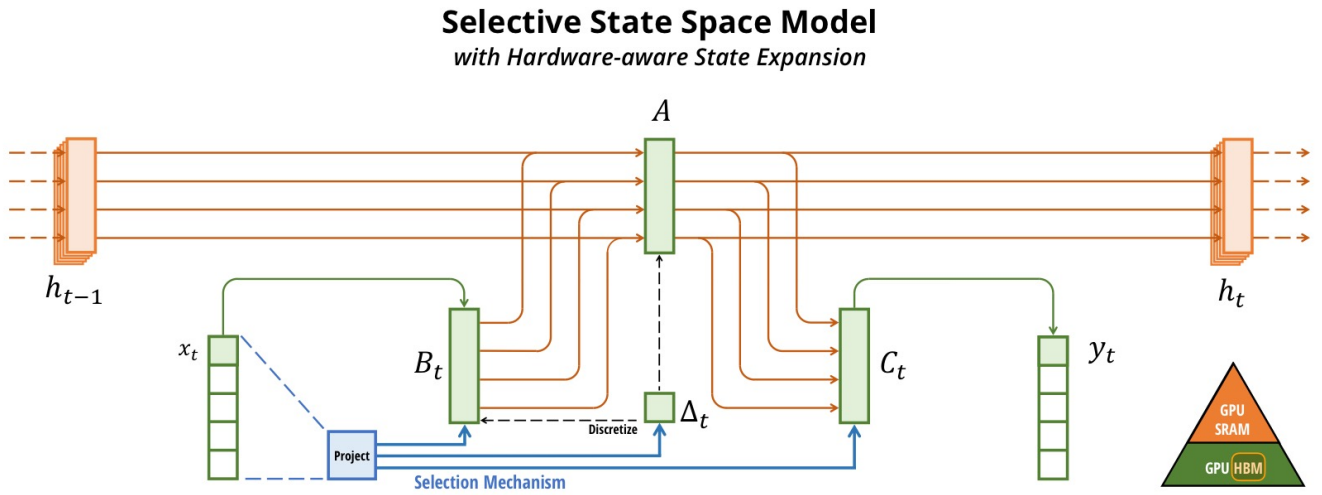


Figure 1: (**Overview**.) Structured SSMs independently map each channel (e.g. $D = 5$) of an input x to output y through a higher dimensional latent state h (e.g. $N = 4$). Prior SSMs avoid materializing this large effective state (DN, times batch size B and sequence length L) through clever alternate computation paths requiring time-invariance: the $(\Delta, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ parameters are constant across time. Our selection mechanism adds back input-dependent dynamics, which also requires a careful hardware-aware algorithm to only materialize the expanded states in more efficient levels of the GPU memory hierarchy.

TRAIN AND TEST

2. Kernel Fusion

3차원 parameter 을
HBM -> SRAM

Discretize 후(4차원) $\bar{A}$와 $\bar{B}$ 를
SRAM-> HBM write

HBM -> SRAM
복사 후
SSM을 계산

SSM 계산 결과인 y를
SRAM-> HBM write

3차원 parameter 을
HBM에 -> SRAM

Discretize를 SRAM에서

SSM 연산 후 3차원인 y를
SRAM->HBM

TRAIN AND TEST

3. recomputation

- 문제 2였던 메모리 사용을 줄이기 위해 **recomputation**
- back propagation이 gradient계산을 위해 필요한 intermediate states 저장(x) ,
- input이 HBM에서 SRAM으로 load될 때 (backward pass)시 intermediate states를 recomputate

⇒ flash attention을 이용한 (optimized)된 transformer와 같은 수준의 메모리 사용이 가능하게 됨

FlashAttention

a. attention layer : 토큰당 12 bytes of activations 를 store
b. MLP layer :토큰당 20 bytes of activations 를 store
c. total : 32 bytes

selective SSM

: 토큰당 16 bytes of activations 를 store

TRAIN AND TEST

*properties of Selection Mechanisms

1) RNN gating (input dependent) **+ discretization of continuous-time systems**(ZOH 이산화)

2) SSM의 Δ는 RNN의 gating mechanism 같은 역할

$$h_t = \overline{A}h_{t-1} + \overline{B}x_t \quad (2a)$$
$$y_t = Ch_t \quad (2b)$$

$$\overline{A} = \exp(\Delta A) \qquad \overline{B} = (\Delta A)^{-1}(\exp(\Delta A) - I) \cdot \Delta B$$

TRAIN AND TEST

*properties of Selection Mechanisms

1) **Variable Spacing** gated RNN 에서 g_t → 0보내는 것과 같이 Selectivity를 통해 um 같은 단어를 filtering 할 수 있음

2) **Filtering Context**

- **general LTI** 의 global convolutions는 irrelevant한 context를 모두 반영

→ longer context가 주어졌을 때 performance improvement(x)

- **Selective model**의 state reset

→ context length에 따라 monotonically improvement (o)

3) Boundary Resetting

- **Transformer**: independent sequence에 각 attention mask를 instantiate하여 seperate하게 유지
- **LTI**: information이 sequence 사이 간 bleed
- **Selective model** : Δ→ ∞ 로 state를 reset할 수 있음

TRAIN AND TEST

*properties of Selection Mechanisms

4) Interpretation of Δ

- 현재 Input을 얼마나 새로운 상태에 반영할지 control
- Δ가 커질수록 현재 input의 영향이 커지고 Δ가 작아질수록 Abar가 1이 되므로 이전 state를 그대로 복사

5) Interpretation of A

- Δ를 통해 A도 selectivity

6) Interpretation of B&C

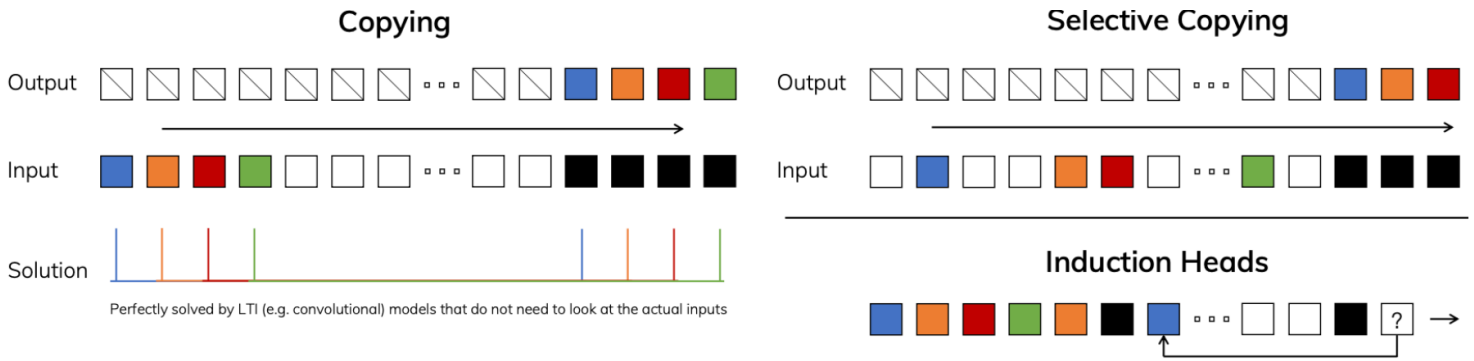
- Δ를 통해 B,C도 selectivity → sequence model의 context가 efficient state로 compressed
- x_t → h_t 혹은 output y_t 에 control 잘 할 수 0
- input과 hidden states각각을 기반으로 recurrent dynamic을 Modulate할 수 있게 함

TRAIN AND TEST

4. Experiment result - Synthetic Task

(1) selective copying

: 회귀 모델의 memorization 성능을 판단
input 토큰간 랜덤한 space를 줌



Model	Arch.	Layer	Acc.
S4	No gate	S4	18.3
-	No gate	S6	**97.0**
H3	H3	S4	57.0
Hyena	H3	Hyena	30.1
-	H3	S6	**99.7**
-	Mamba	S4	56.4
-	Mamba	Hyena	28.4
Mamba	Mamba	S6	**99.8**

Table 1: (**Selective Copying**.)
Accuracy for combinations of architectures and inner sequence layers.

기존 LTI SSMs ⇒ input 길이에 딱 맞는 global convolutions를 이용 : 추론 x
선행연구: 그럼 gating를 줘서 모델에 data-dependence를 부여해서 해결가능?

저자 : gating 은 position aware x

TRAIN AND TEST

4. Experiment result - Synthetic Task

(2) induction heads : in-context learning

- associative recall and copy를 요구.
- A B sequence를 이전에 보았고, A가 다시 등장했을 때 B를 ㅋ

Data

- train - 2-layer 모델로 256 length sequence
- eval - 2^6 ~ 2^20

Model

layer 2

8heads인 multi-head attention과 SSM 변수에 test

- Mamaba- 64d
- 다른 모델: 128d

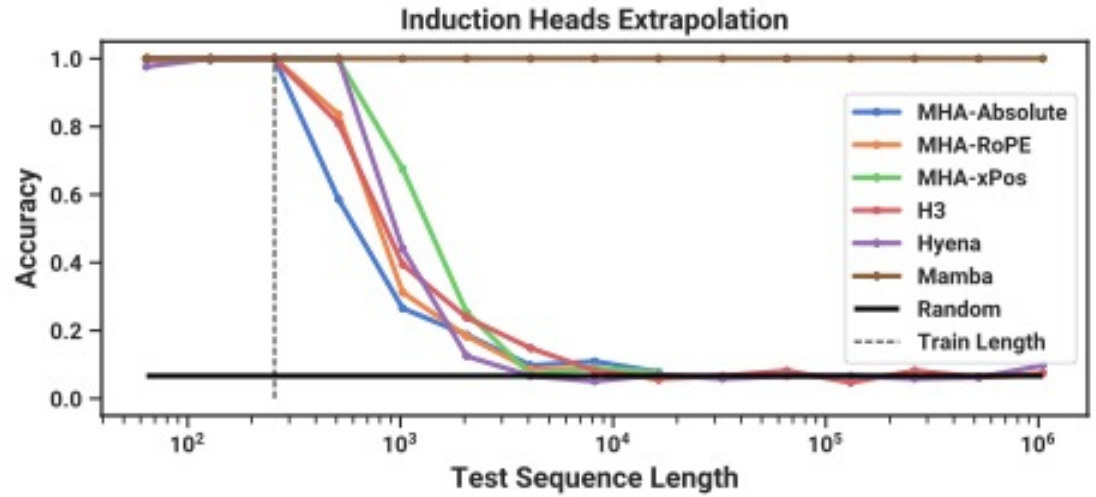


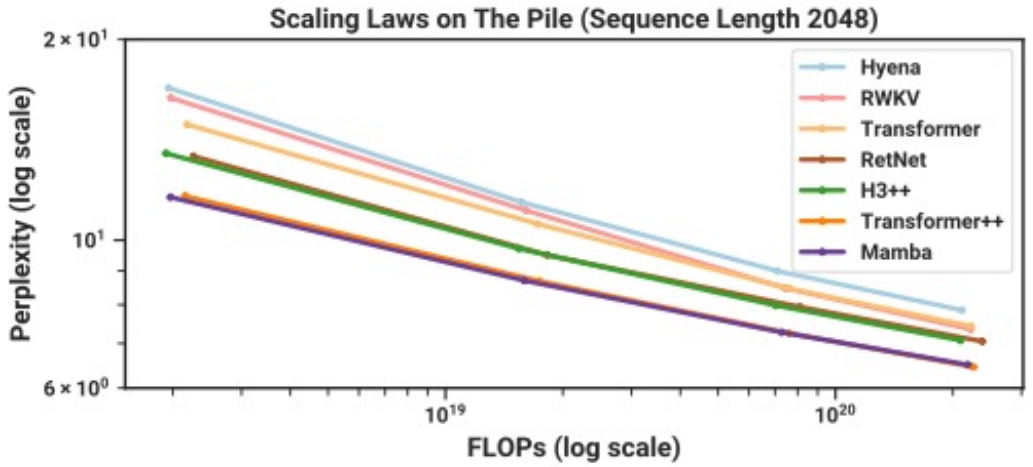
Table 2: (**Induction Heads.**) Models are trained on sequence length $2^8 = 256$, and tested on increasing sequence lengths of $2^6 = 64$ up to $2^{20} = 1048576$. Full numbers in Table 11.

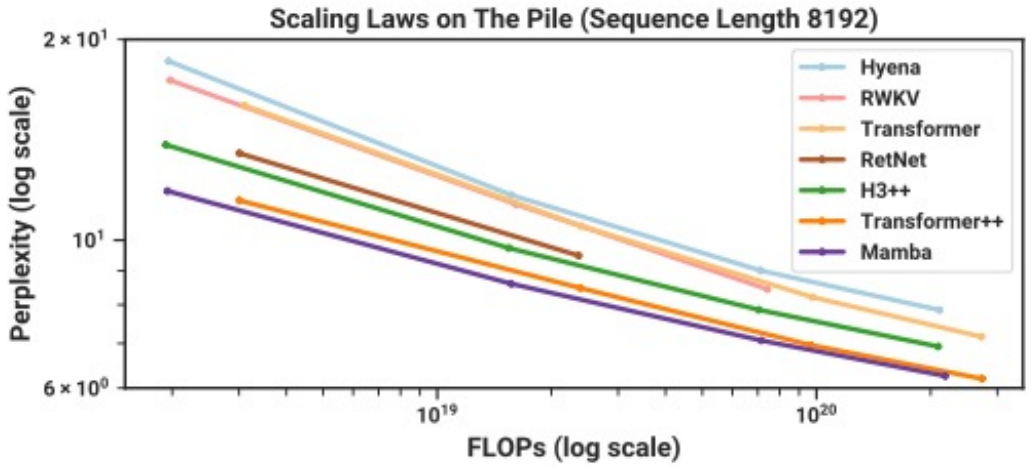
TRAIN AND TEST

4. Experiment result
- language model pretraining(scaling law), zero-shot/perplexity

scaling law: FLOPs(컴퓨터의 성능을 수치로 나타낼 때 주로 사용되는 단위)커질 수록 ppl 낮아짐

Mamba : attention-free 모델에 비해 transformer ++만큼 scaling law를 잘 따름





TRAIN AND TEST

4. Experiment result - zero-shot

Table 3: (**Zero-shot Evaluations**.) Best results for each size in bold. We compare against open source LMs with various tokenizers, trained for up to 300B tokens. Pile refers to the validation split, comparing only against models trained on the same dataset and tokenizer (GPT-NeoX-20B). For each model size, Mamba is best-in-class on every single evaluation result, and generally matches baselines at twice the model size.

Model	Token.	Pile ppl ↓	LAMBADA ppl ↓	LAMBADA acc ↑	HellaSwag acc ↑	PIQA acc ↑	Arc-E acc ↑	Arc-C acc ↑	WinoGrande acc ↑	Average acc ↑
Hybrid H3-130M	GPT2	—	89.48	25.77	31.7	64.2	44.4	24.2	50.6	40.1
Pythia-160M	NeoX	29.64	38.10	33.0	30.2	61.4	43.2	24.1	**51.9**	40.6
Mamba-130M	NeoX	**10.56**	**16.07**	**44.3**	**35.3**	**64.5**	**48.0**	**24.3**	**51.9**	**44.7**
Hybrid H3-360M	GPT2	—	12.58	48.0	41.5	68.1	51.4	24.7	54.1	48.0
Pythia-410M	NeoX	9.95	10.84	51.4	40.6	66.9	52.1	24.6	53.8	48.2
Mamba-370M	NeoX	**8.28**	**8.14**	**55.6**	**46.5**	**69.5**	**55.1**	**28.0**	**55.3**	**50.0**
Pythia-1B	NeoX	7.82	7.92	56.1	47.2	70.7	57.0	27.1	53.5	51.9
Mamba-790M	NeoX	**7.33**	**6.02**	**62.7**	**55.1**	**72.1**	**61.2**	**29.5**	**56.1**	**57.1**
GPT-Neo 1.3B	GPT2	—	7.50	57.2	48.9	71.1	56.2	25.9	54.9	52.4
Hybrid H3-1.3B	GPT2	—	11.25	49.6	52.6	71.3	59.2	28.1	56.9	53.0
OPT-1.3B	OPT	—	6.64	58.0	53.7	72.4	56.7	29.6	59.5	55.0
Pythia-1.4B	NeoX	7.51	6.08	61.7	52.1	71.0	60.5	28.5	57.2	55.2
RWKV-1.5B	NeoX	7.70	7.04	56.4	52.5	72.4	60.5	29.4	54.6	54.3
Mamba-1.4B	NeoX	**6.80**	**5.04**	**64.9**	**59.1**	**74.2**	**65.5**	**32.8**	**61.5**	**59.7**
GPT-Neo 2.7B	GPT2	—	5.63	62.2	55.8	72.1	61.1	30.2	57.6	56.5
Hybrid H3-2.7B	GPT2	—	7.92	55.7	59.7	73.3	65.6	32.3	61.4	58.0
OPT-2.7B	OPT	—	5.12	63.6	60.6	74.8	60.8	31.3	61.0	58.7
Pythia-2.8B	NeoX	6.73	5.04	64.7	59.3	74.0	64.1	32.9	59.7	59.1
RWKV-3B	NeoX	7.00	5.24	63.9	59.6	73.7	67.8	33.1	59.6	59.6
Mamba-2.8B	NeoX	**6.22**	**4.23**	**69.2**	**66.1**	**75.2**	**69.7**	**36.3**	**63.5**	**63.3**
GPT-J-6B	GPT2	–	4.10	68.3	66.3	75.4	67.0	36.6	64.1	63.0
OPT-6.7B	OPT	–	4.25	67.7	67.2	76.3	65.6	34.9	65.5	62.9
Pythia-6.9B	NeoX	6.51	4.45	67.1	64.0	75.2	67.3	35.5	61.3	61.7
RWKV-7.4B	NeoX	6.31	4.38	67.2	65.5	76.1	67.8	37.5	61.0	62.5

TRAIN AND TEST

4. Experiment result - DNA sequence pre-training

DNA

(1) 유한한 vocab, discrete된 sequence로 구성됐다는 점에서 언어와 유사

(2) long-range dependencies도 필요

Model

hyenaDNA

DATA

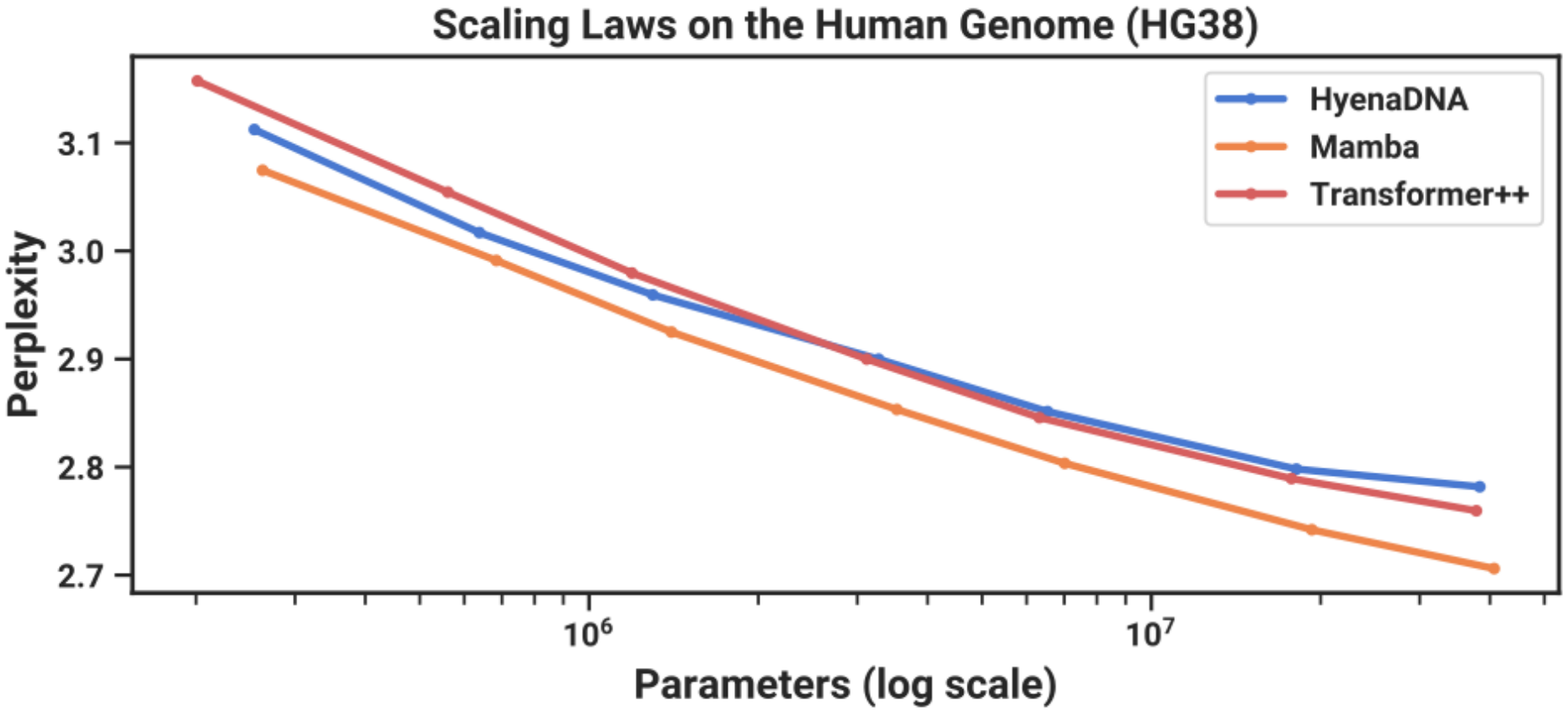
HG38 데이터셋

-> pretraining : single human genome, 약 45억 개의 토큰(DNA 염기쌍)

TRAIN AND TEST

4. Experiment result - DNA sequence pre-training

(1) Model size scaling



TRAIN AND TEST

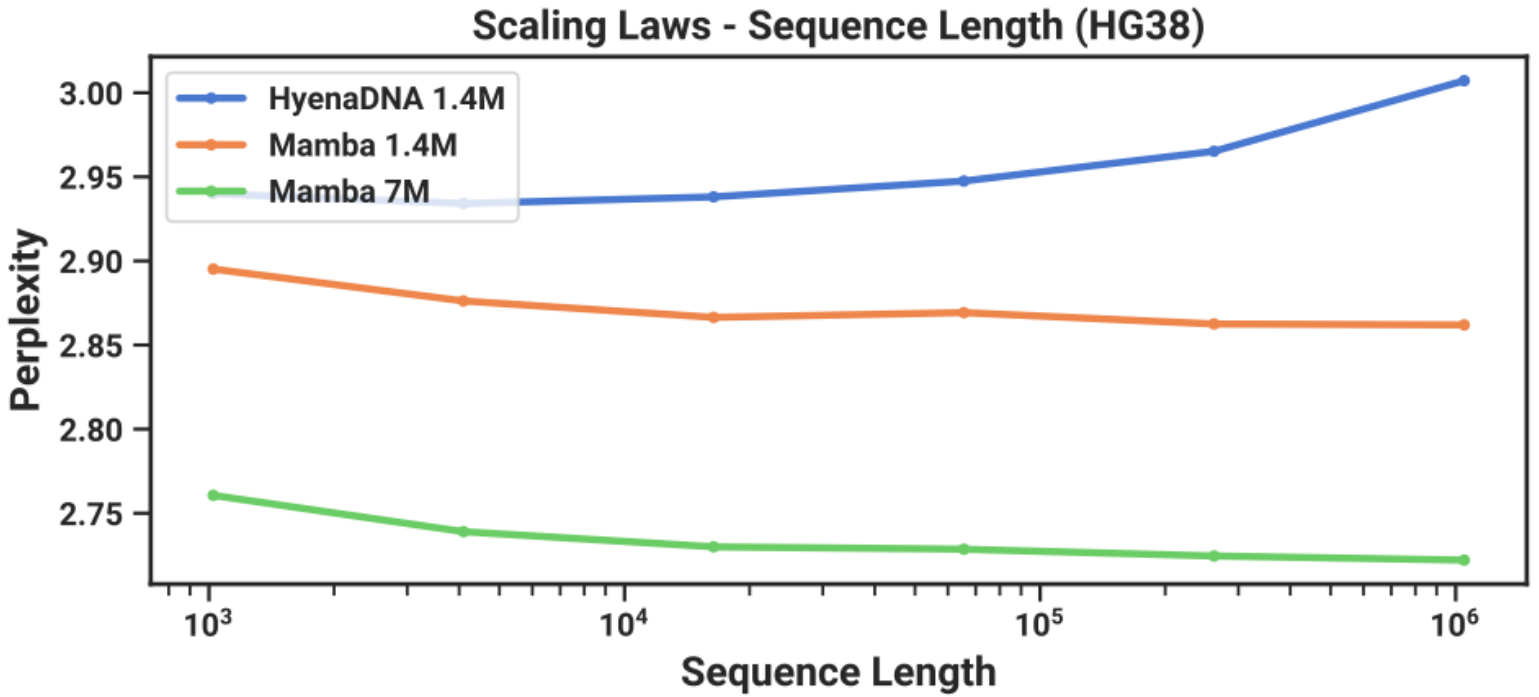
4. Experiment result - DNA sequence pre-training

(2) Context length scaling

HyenaDNA vs. Mamba

(∵ Attention은 긴 시퀀스에서는 매우 비용이 많이 듦)

MAMBA: 긴 시퀀스 길이에 대해서도 perplexity 성능 유지

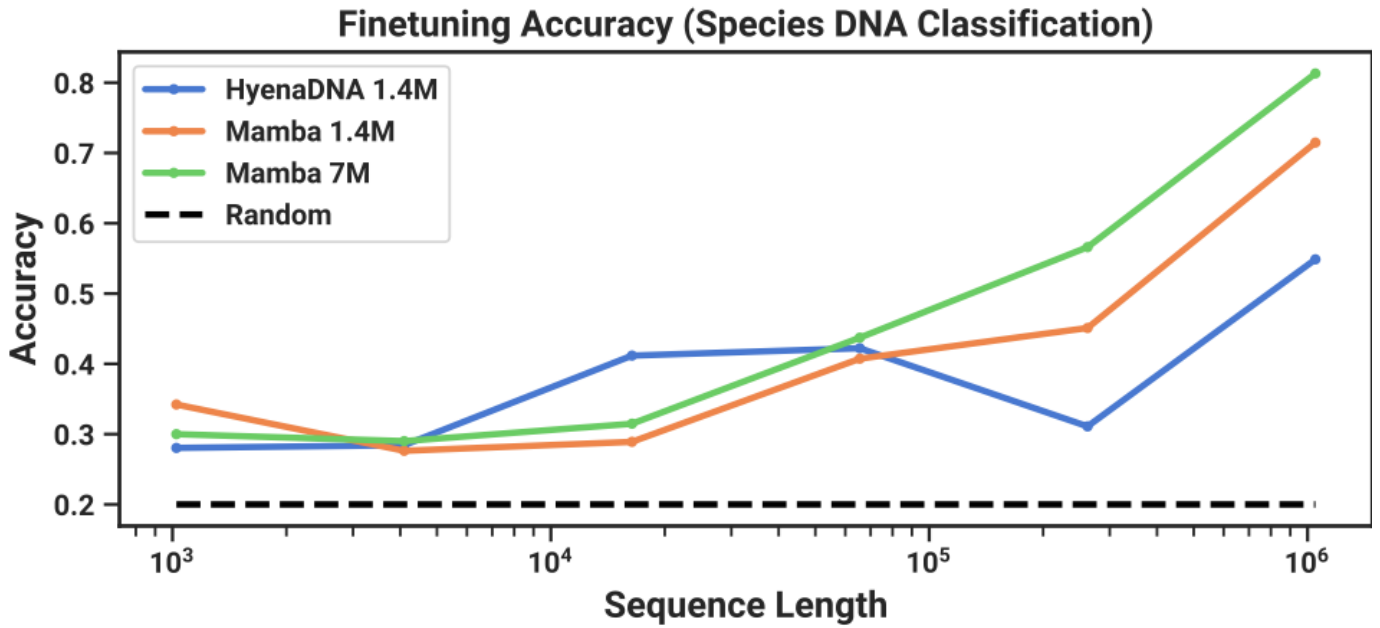


TRAIN AND TEST

4. Experiment result - DNA sequence pre-training

(3) synthetic species classification

downstream task: 5가지 다른 종을 분류
{human, lemur, mouse, pig, hippo}
더 어려운 task: {human, chimpanzee, gorilla, orangutan, bonobo}: DNA 95%이상 공유



TRAIN AND TEST

4. Experiment result - audio waveform pretraining

DATA

SC09 benchmark

- speech generation dataset
- autoregressively generated speech clips dataset

Model	Params	NLL ↓	FID ↓	IS ↑	mIS ↑	AM ↓
SampleRNN	35.0M	2.042	8.96	1.71	3.02	1.76
WaveNet	4.2M	1.925	5.08	2.27	5.80	1.47
SaShiMi	5.8M	1.873	1.99	5.13	42.57	0.74
WaveGAN	19.1M	-	2.03	4.90	36.10	0.80
DiffWave	24.1M	-	1.92	5.26	51.21	0.68
+ SaShiMi	23.0M	-	1.42	5.94	69.17	0.59
Mamba	6.1M	**1.852**	0.94	6.26	88.54	0.52
Mamba	24.3M	1.860	**0.67**	**7.33**	**144.9**	**0.36**
Train	-	-	0.00	8.56	292.5	0.16
Test	-	-	0.02	8.33	257.6	0.19

GAN / Diffusion model < small mamba

TRAIN AND TEST

4. Experiment result - Speed and Memory Benchmarks

a. 훈련속도: parallel scan, recomputation, kernel fusion

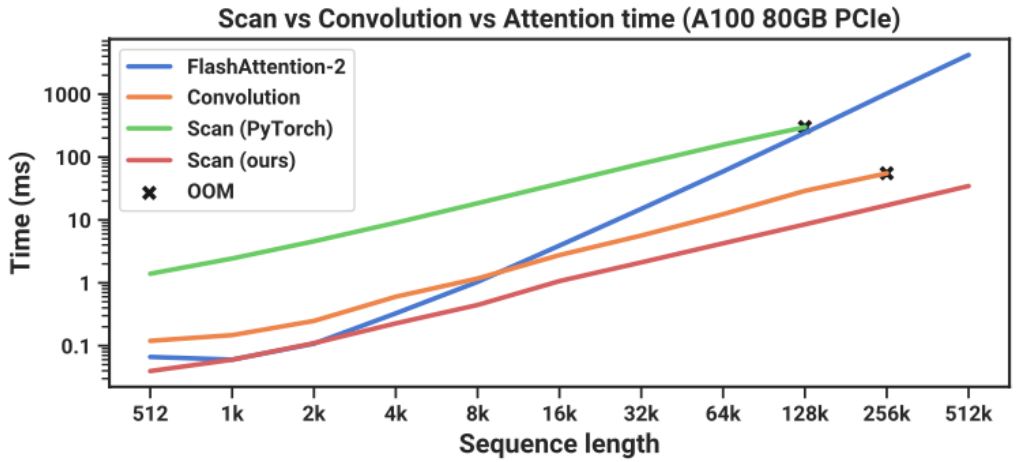
→ 2k이상 시퀀스에 대해서 FlashAttention2보다도 빠름

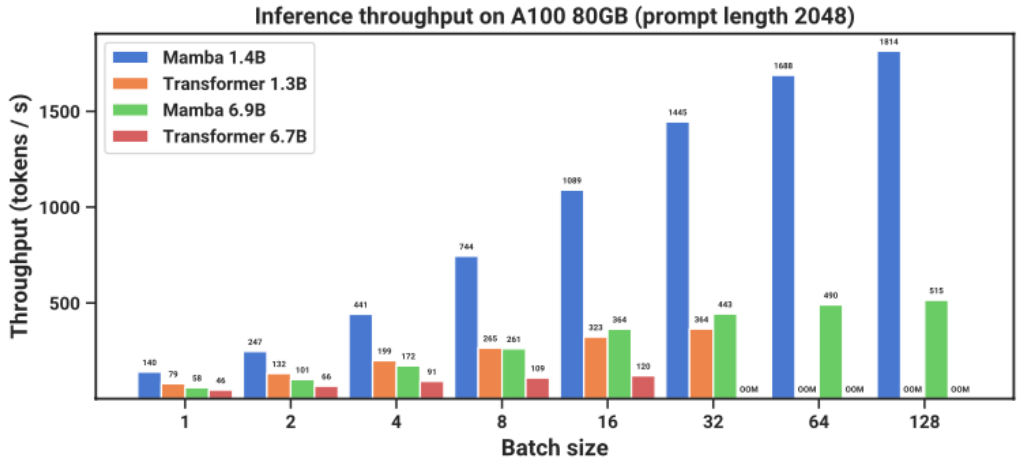
→ Hardware-aware scan이 기본 scan (pytorch)보다도 빠름 20-40배 빠름

b. inference time 측면

→ 원래 attention-free가 갖는 장점

→ 월등히 빠른 속도 : Mamba 6.9B가 Transformer 1.3B와 비슷한 정도





TRAIN AND TEST

4. Experiment result - components ablate

(1) Architecture: block Architecture 와 SSM layer의 효과

a. LTI SSM(비선택적, global convolution) 성능 비슷

b. complex-valued S4와 real-valued S4의 차이는 성능에 영향 x

: 그렇다면 하드웨어 관점에서 real-valued를 사용하는 것이 좋음

c. S6로 교체 시 성능 향상

d. H3 구조와 성능 비슷

Model	Arch.	SSM Layer	Perplexity
Hyena	H3	Hyena	10.24
H3	H3	S4 (complex)	10.30
-	H3	S4 (real)	10.34
-	H3	S6	**8.95**

Model	Arch.	SSM Layer	Perplexity
-	Mamba	Hyena	10.75
-	Mamba	S4 (complex)	10.54
-	Mamba	S4 (real)	10.56
Mamba	Mamba	S6	**8.69**

TRAIN AND TEST

4. Experiment result - components ablate

(2) selective SSM : selective Δ, B 및 C 매개변수의 효과

a. Δ

- Δ는 RNN gating 모델의 유사성 때문에 selective를 가능하게 한다는 점에서 매우 중요

Selective Δ	Selective B	Selective C	Perplexity
✗	✗	✗	10.93
✗	✓	✗	10.15
✗	✗	✓	9.98
✓	✗	✗	9.81
✓	✓	✓	8.71

TRAIN AND TEST

4. Experiment result - components ablate

(2) selective SSM : selective Δ, B 및 C 매개변수의 효과

b. A

simpler real-valued diagonal initialization > standard complex-valued parameterizations

Table 8: (**Ablations: Parameterization of A.**) The more standard initializations based on S4D-Lin (Gu, Gupta, et al. 2022) perform worse than S4D-Real or a random initialization, when the SSM is selective.

A_n Initialization	Field	Perplexity
$A_n = -\frac{1}{2} + ni$	Complex	9.16
$A_n = -1/2$	Real	8.85
$A_n = -(n+1)$	Real	8.71
$A_n \sim \exp(\mathcal{N}(0,1))$	Real	8.71

TRAIN AND TEST

4. Experiment result - components ablate

(2) selective SSM : selective Δ, B 및 C 매개변수의 효과

c. Δ, B 및 C 의 차원에 변화

- static → selective 변화 : 가장 benefit

- 차원이 커질수록 매개변수가 조금 늘어나면서도 성능이 향상
  특히 state size N을 증가 시킬 때, parameter cost가 1%만 올랐는데 perplexity가 1 이상 향상

Table 9: (**Ablations: Expressivity of Δ.**) The selection mechanism of Δ constructs it with a projection of the input. Projecting it even to dim. 1 provides a large increase in performance; increasing it further provides further improvements at the cost of a modest increase in parameters. State size fixed to $N = 16$.

Size of Δ proj.	Params (M)	Perplexity
-	358.9	9.12
1	359.1	8.97
2	359.3	8.97
4	359.7	8.91
8	360.5	8.83
16	362.1	8.84
32	365.2	8.80
64	371.5	8.71

Table 10: (**Ablations: SSM state dimension.**) (*Top*) Constant $\boldsymbol{B}$ and $\boldsymbol{C}$ (*Bottom*) Selective $\boldsymbol{B}$ and $\boldsymbol{C}$. Increasing the SSM state dimension N, which can be viewed as an expansion factor on the dimension of the recurrent state, can significantly improve performance for a negligible cost in parameters/FLOPs, but only when $\boldsymbol{B}$ and $\boldsymbol{C}$ are also selective. Size of Δ projection fixed to 64.

State dimension N	Params (M)	Perplexity
1	367.1	9.88
2	367.4	9.86
4	368.0	9.82
8	369.1	9.82
16	371.5	9.81
1	367.1	9.73
2	367.4	9.40
4	368.0	9.09
8	369.1	8.84
16	371.5	8.71

5. Comparison With RNN and Transformer

Transformer

- Attention
- Parallel -> fast training
- 소모적 search => high cost & slow inference

RNN

- Step t-1 -> step t
  =>hidden state의 parallelize (x)

- 이전 state를 메모리에 모두 저장
  => low cost & fast inference

MAMBA

- RNN 계승

메모리 저장 단점 보완: 잘하기 위한 방법으로 **structure SSM**사용
병렬화 단점 보완: **parallel scan**

TRAIN AND TEST

5. Comparison With RNN and Transformer

Comparison to Transformers [edit]

Feature	Transformer	Mamba
Architecture	Attention-based	SSM-based
Complexity	High	Lower
Inference Speed	$O(n)$	$O(1)$
Training Speed	$O(n^2)$	$O(n)$

TRAIN AND TEST

TRAIN AND TEST