Paper You Should Read

Mamba: Linear-Time Sequence Modeling with Selective State Spaces

Albert Gu, Tri Dao

arXiv 2023 / 12 / 1 (# of citations: 129)

고려대학교 산업경영공학과

DSBA 연구실

천재원

Mamba

Contents

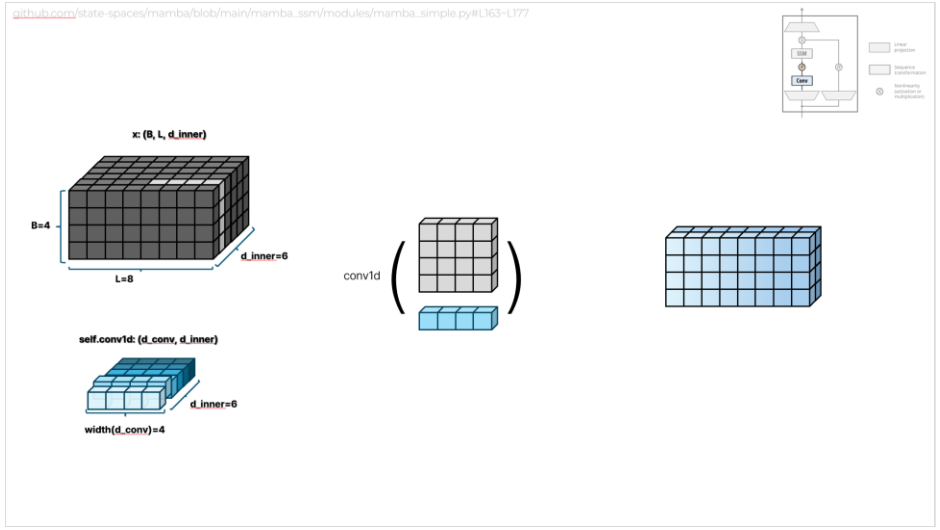
Mamba

Contents

Contents

Contents

Contents



In the next video(If there's a need...)

Mamba

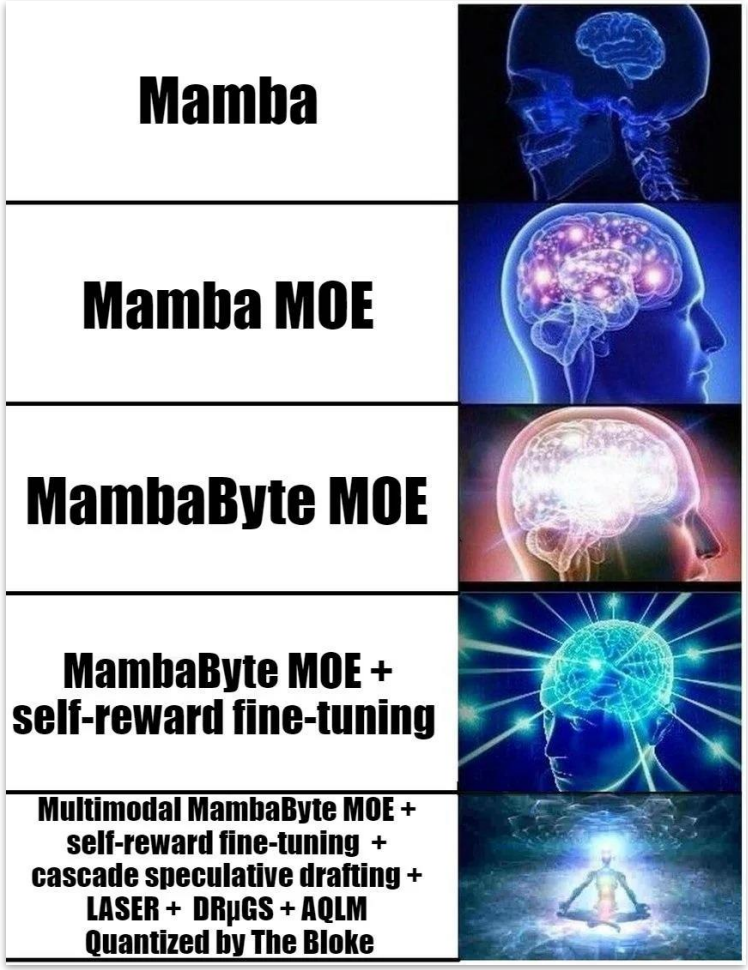
Contents

Preliminary

Preliminary

https://www.reddit.com/r/LocalLLaMA/comments/19fgpvy/llm_enlightenment/

- Mamba is…
  - ✓ Dealing with Sequential Data
    - Given $X_{1:T-1}$ → Predict X_T를 위한 모델
  - ✓ Deeply stacked blocks of State-Space Models(SSM)
    - 시간에 따라 변화하는 신호를 모델링하는 하나의 방법
  - ✓ Now growing its own Ecosystem



Preliminary

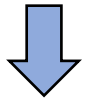
- Why is it needed?

LSTM is dead, Long live Transformer!

Transformer는 꽤 오래 전부터 딥러닝 아키텍처의 헤게모니를 차지해 왔음

뛰어난 성능

빠른 훈련 속도

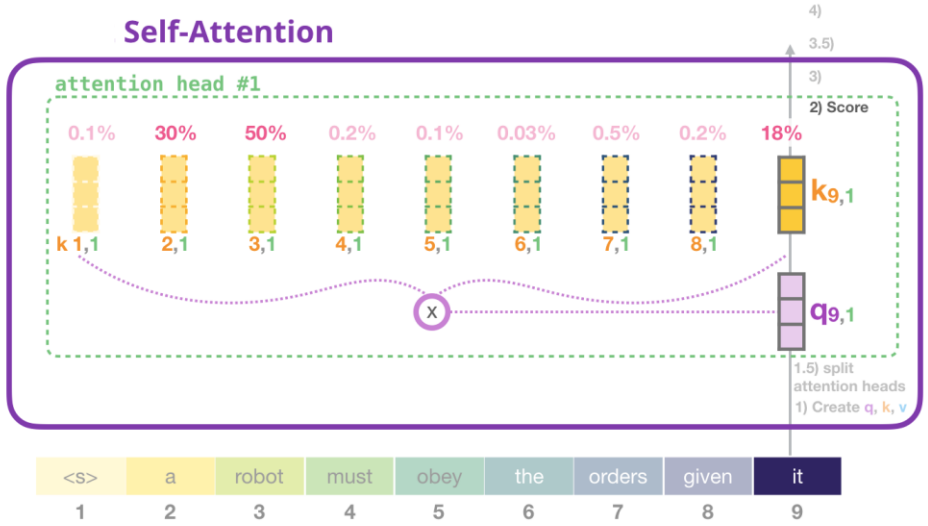


역으로, Transformer가 아닌 다른 아키텍처를 연구하는 사람들은 어떤 부분을 파고 드는가?

Mamba

Preliminary

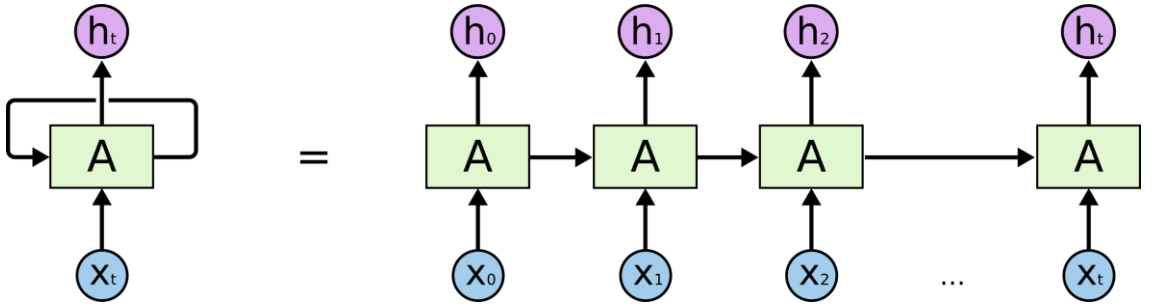
- Why is it needed?
  - ✓ Attention enables Training Parallelism, in turn, Fast Training!
    - 이전 Time-step에 대한 모델의 예측 값은 다음 Time-step에 대한 모델의 예측 값과 상관 X
    - 한 번의 forward로 모든 Time-step에 대한 훈련이 가능 ← Training Parallelism
  - ✓ Attention forces Exhaustive Search, in turn, High Cost & Slow Inference!
    - Inference시, 하나의 Token을 생성하는 데에 걸리는 시간이 Linear하게 늘어남
    - Attend 해야 하는 states들의 수 또한 Linear하게 늘어남



Mamba

Preliminary

- Why is it needed?
  - ✓ Memory's update cannot be **Parallelized**, in turn, **Slow Training!**
    - 이전 Time-step에 대한 Memory(hidden state)가 다음 Time-step의 Input으로 필요
    - 한 번의 forward로 한 번의 Memory update와 하나의 output이 나오고, 그 Memory를 다음 forward에 넣어주고...
  - ✓ Memory carries total information, in turn, **Low Cost & Fast Inference!**
    - 지금 시점의 output은 바로 이전 시점에서 넘어온 memory와 지금 시점의 input에만 의존함
    - 토큰 당 생성 속도 및 VRAM 요구치도 Constant



Mamba

Preliminary

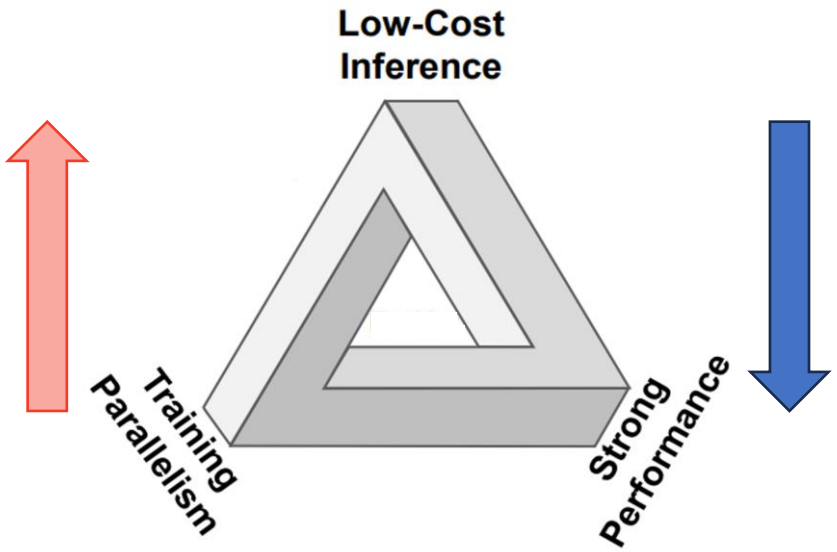
- Why is it needed?

Transformers

Towards Low Cost

RNNs

Towards Performance + Parallel Train



Preliminary

- Why is it needed?

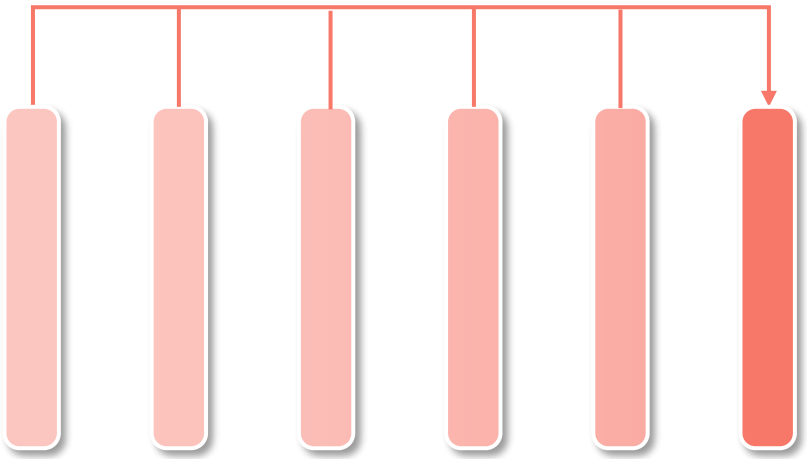
Transformers

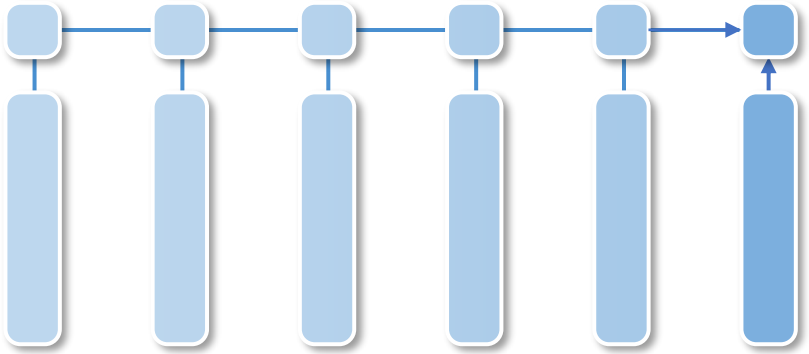
Towards Low Cost

RNNs

Towards Performance + Parallel Train

Naturally arising limitation as of their architecture (Attention vs Memory)





Mamba

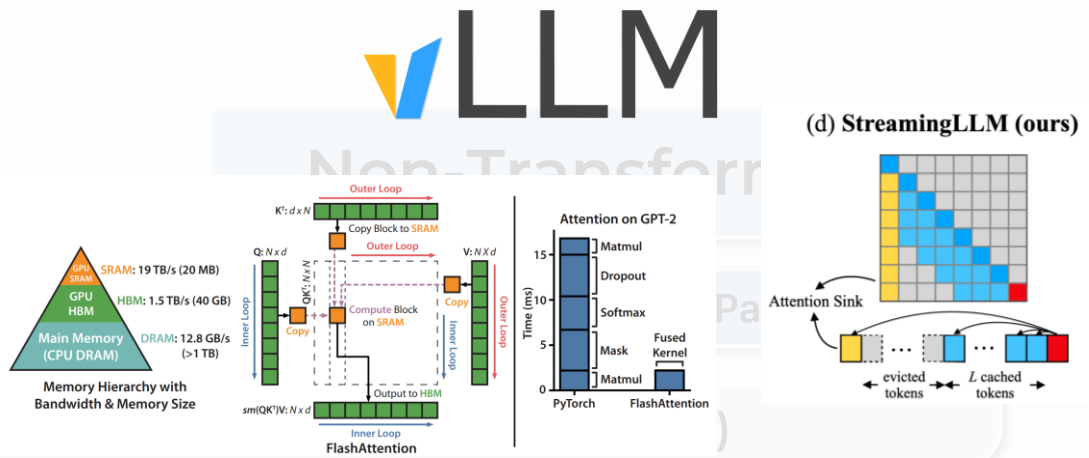
Preliminary

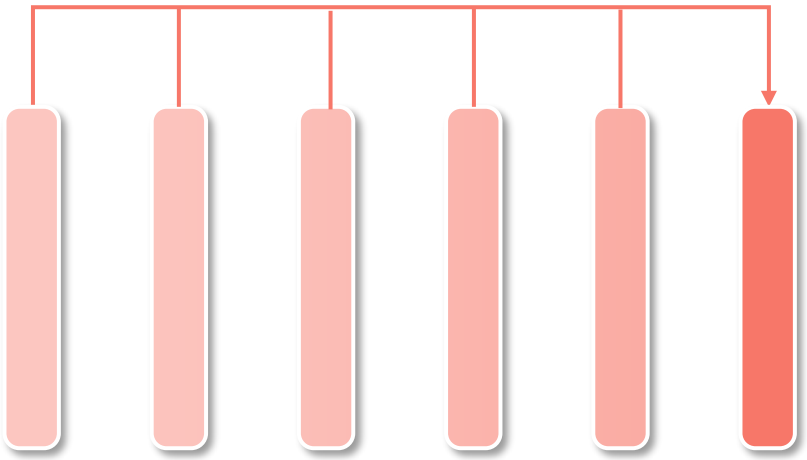
- Why is it needed?

Transformers

Towards Low Cost

Naturally arising limitation as of their ar





굳이 왜 sequence 전체의 정보를 모아서 저장해 둬야 하지?

그냥 각각의 정보에 내가 알아서 Attend하고 종합할게!

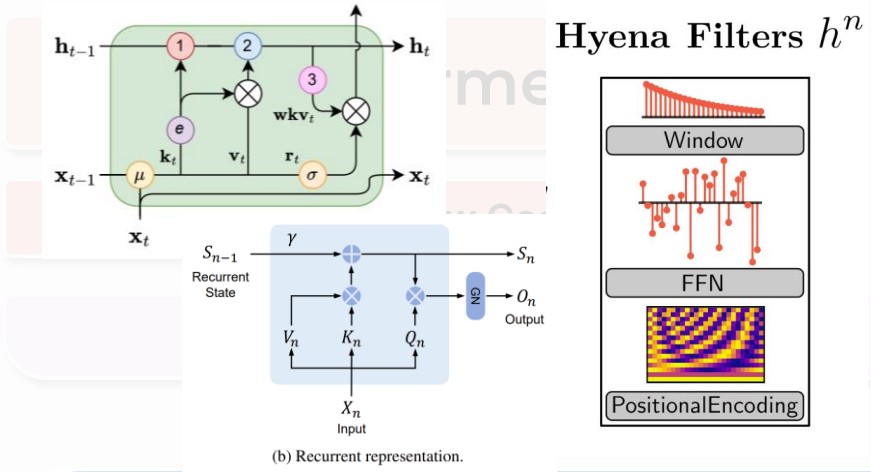
(...근데 그럼 길이가 늘어날수록 느려지긴 할텐데)

(...메모리 엄청 잡아먹긴 하는데)

Mamba

Preliminary

- Why is it needed?



(b) Recurrent representation.

RNNs

Towards Performance + Parallel Train

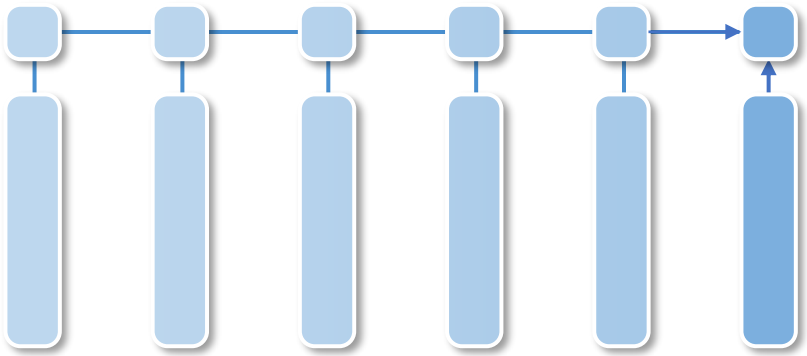
ir architecture (Attention vs Memory)

그렇게 하나하나 언제 다 보고 있음?

Memory에 저장만 잘 해 놓으면 되는데?

(...한정된 공간에 어떻게 잘 저장하지)

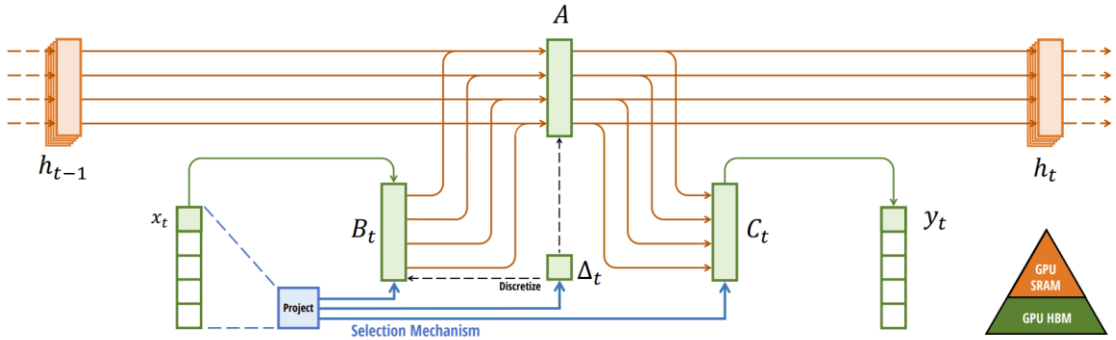
(...훈련은 어떻게 좀 빠르게 안 되나)



Mamba

Preliminary

- Why is it needed?



그렇게 하나하나 언제 다 보고 있음?

Memory에 저장만 잘 해 놓으면 되는데?

Structured SSM 쓰니까 Memory에 다 들어가던데?

Parallel scan으로 훈련도 Transformer보다 빠른데?

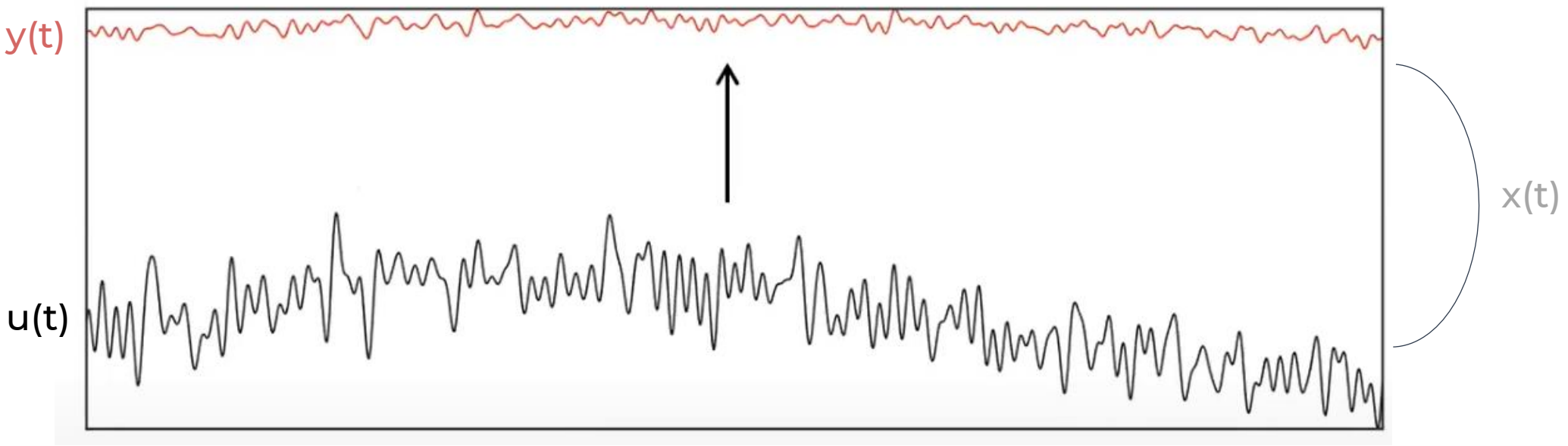
Non-Transformer 계열의 Hottest 모델, Mamba

→ 단순한 Sequence뿐만 아니라,
LLM에도 적용할 수 있음을 보였던 모델

Mamba

Preliminary

- What is that key ‘smart’, SSM?
  - ✓ SSM이 무엇인지 이해가 되는가? → O
  - ✓ 왜 SSM을 쓰는가? → X
    - SSM을 사용하여 Sequence를 모델링 한다는 것이 무슨 뜻인가?



✓ Mapping of 1-D input signal u(t) to N-D latent space x(t), in turn, projecting it back to 1-D y(t)

Preliminary

- What is that key ‘smart’, SSM?
  - ✓ Let’s think in more ‘vector-like’ way!
  - ✓ A와 B는 Memory update에 관여
    - A는 현재 Memory 상태를 transform, B는 현재 input을 transform
    - 만들어진 두 벡터의 합 → Memory의 변화율
  - ✓ C와 D는 Output projection에 관여
    - C는 현재 Memory로부터, D는 현재 input으로부터 값을 받아 옴
    - 만들어진 두 스칼라의 합 → output
    - D는 생략 가능

$$\mathbf{y}(t) \in \mathbb{R}^1$$
$$\mathbf{x}(t) \in \mathbb{R}^n$$
$$\mathbf{u}(t) \in \mathbb{R}^1$$

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

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

Preliminary

- What is that key ‘smart’, SSM?
  - ✓ Let’s think in more ‘discretized’ way!
  - ✓ 아직 이걸로 뭘 할 수 있는지 와닿지 않을 수 있음
    - 흔히 다룰 수 있는 discrete한 sequence에 적용할 수 있게 바꿔보자!
  - ✓ 이런 ODE를 푸는 방법에는 여러가지가 있지만, 예를 들어 Euler method를 통해 풀어보면…

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

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

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

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

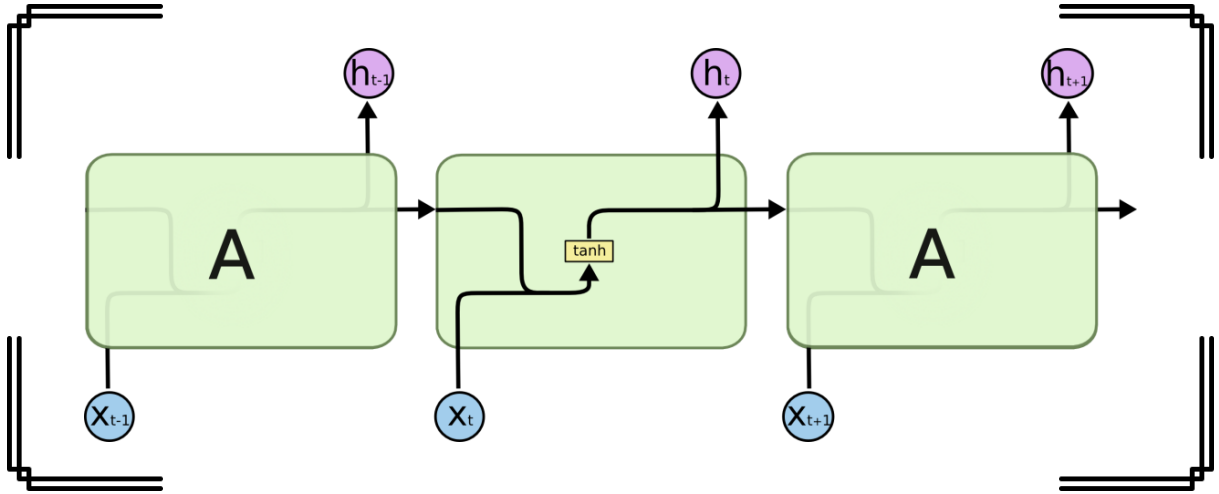
Preliminary

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

- What is that key 'smart', SSM?
  - ✓ 이렇게 Discretize를 하고 나면, RNN과 거의 똑같은 식이 나옴
    - h_t라는 memory가 sequence 전반으로 흘러감
    - 이 memory가 x_t와 합쳐져서 다음 memory를 구성하게 됨

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

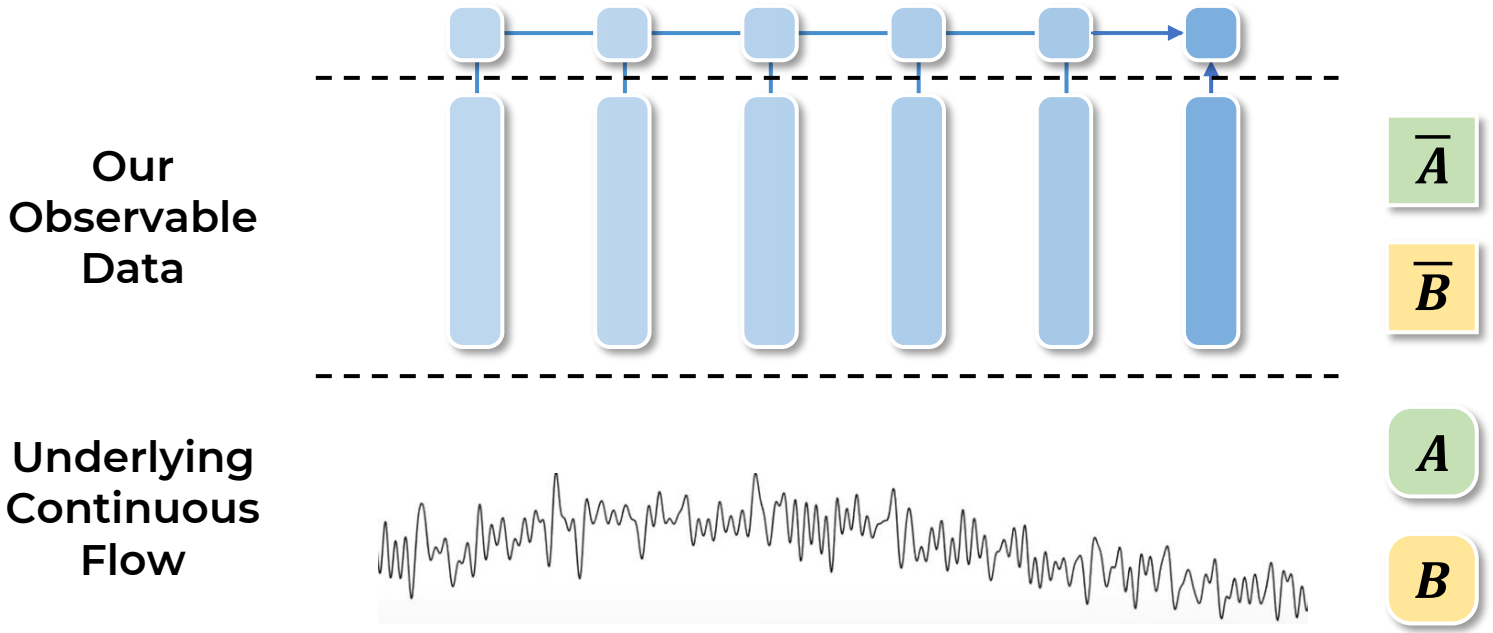
?



그럼 RNN이랑 뭐가 다르죠?

Preliminary

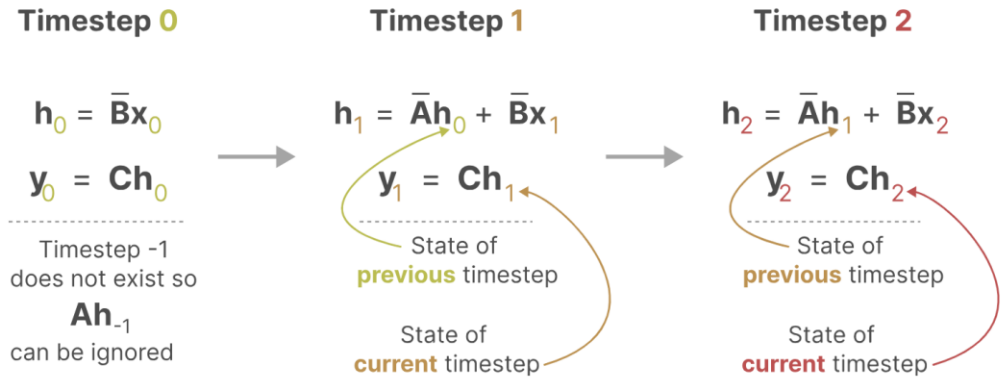
- What is that key ‘smart’, SSM?
  - ✓ It is still continuous!
    - RNN과 똑같은 원리로 discrete domain에서도 쓸 수 있는 것은 맞음
    - 우리가 가진(훈련시킬) 파라미터는 도함수를 운용하는 A, B이지, $\overline{A}, \overline{B}$가 아님

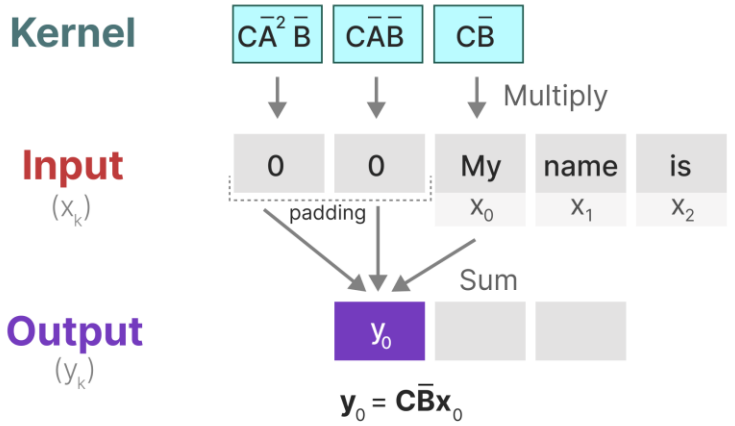


Mamba

Preliminary

- What is that key 'smart', SSM?
  - ✓ It is LTI System!
    - Non-linearity가 없음 → Fast Training이 가능
    - Recurrence를 Convolutionize할 수 있음

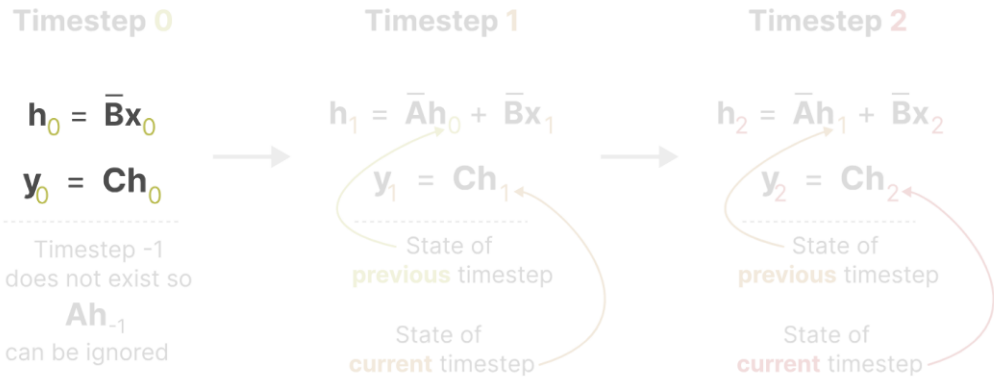


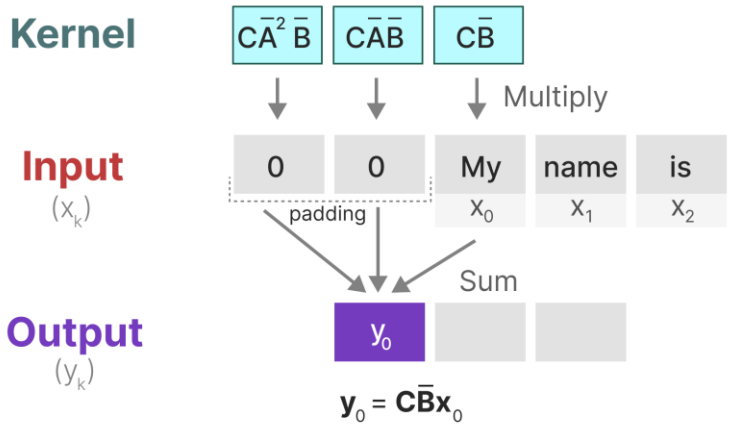


Mamba

Preliminary

- What is that key 'smart', SSM?
  - ✓ It is LTI System!
    - Non-linearity가 없음 → Fast Training이 가능
    - Recurrence를 Convolutionize할 수 있음

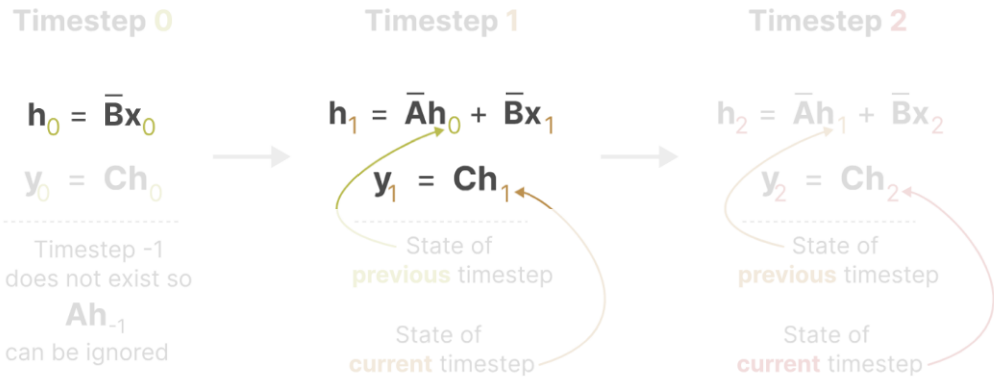


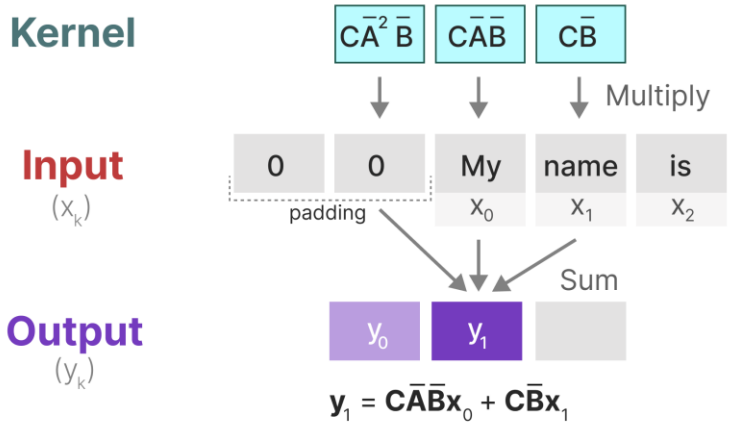


Mamba

Preliminary

- What is that key 'smart', SSM?
  - ✓ It is LTI System!
    - Non-linearity가 없음 → Fast Training이 가능
    - Recurrence를 Convolutionize할 수 있음

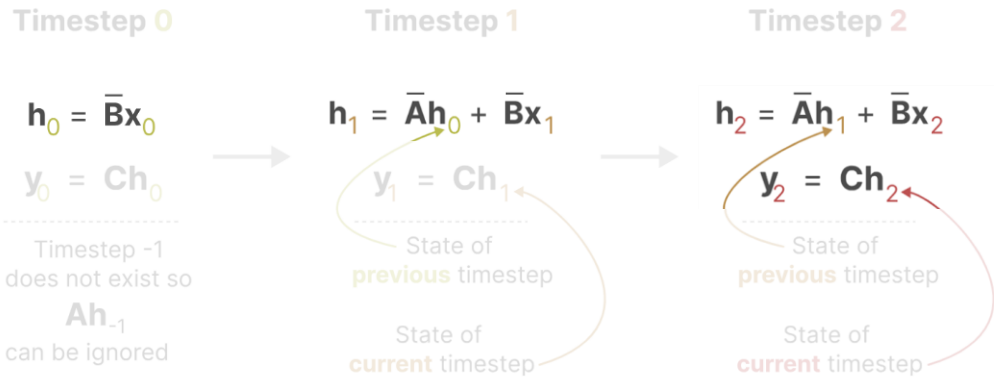


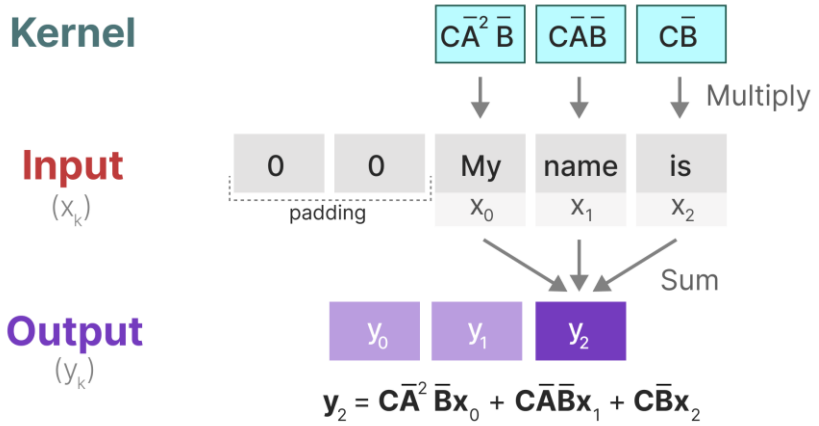


Mamba

Preliminary

- What is that key 'smart', SSM?
  - ✓ It is LTI System!
    - Non-linearity가 없음 → Fast Training이 가능
    - Recurrence를 Convolutionize할 수 있음



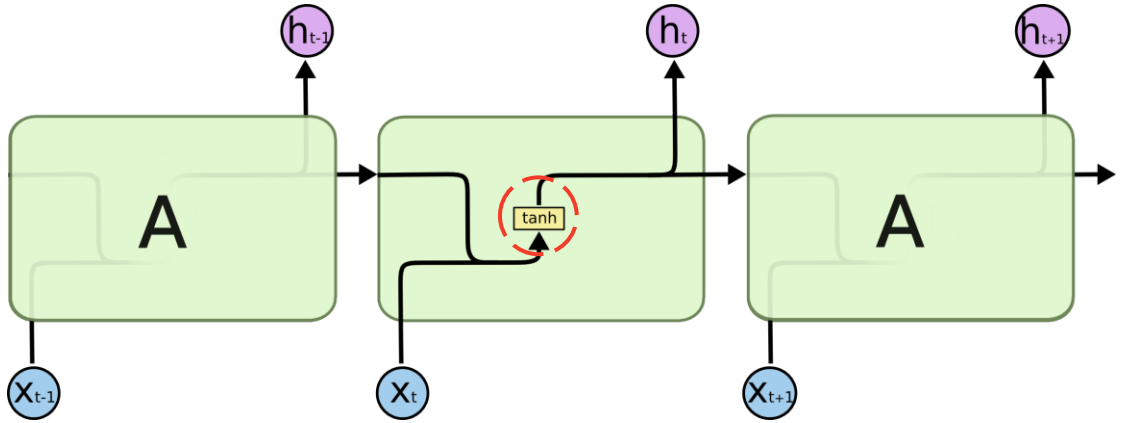


Preliminary

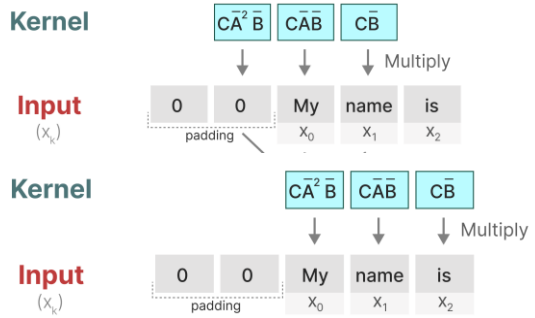
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

- What is that key 'smart', SSM?
  - ✓ Why not RNN? 🥶 🥶 🥶
    - Activation
    - Time-variant
    - Data-dependent

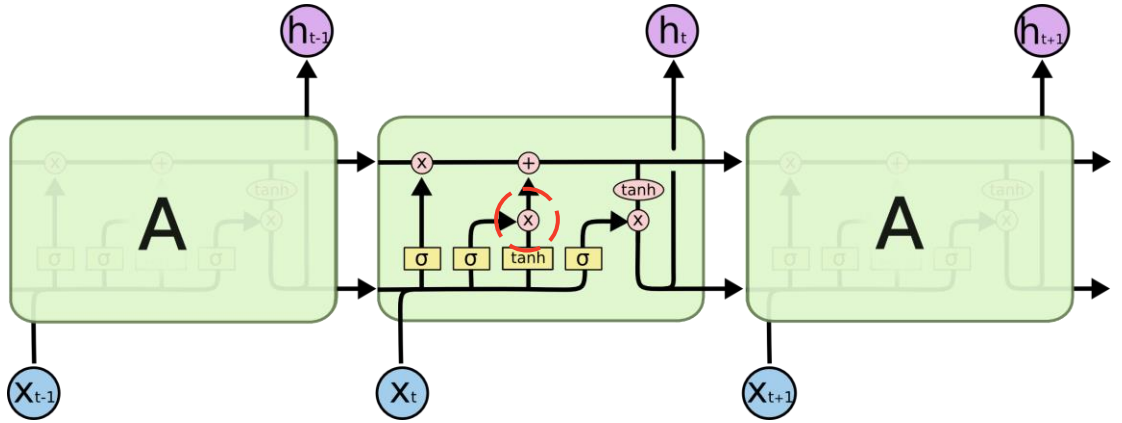
[RNN]



[SSM]



[LSTM]



Preliminary

- What is that key ‘smart’, SSM?

SSM, Good-old Method for Continuous Function

Discretization을 통해, 우리가 가지고 있는 여러 discrete한 sequence를 다룰 수 있음

Updating Memory | **Convey Input** | **Project to Output**

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

$$\mathbf{y}_t = \mathbf{C}\mathbf{x}_t$$

Fast training with Conv mode, Light inference with Recur mode, (Performance...?)

Towards Mamba!

Towards Mamba!

Mamba

Towards Mamba!

- LSSL

✓ SSM을 사용하여 Sequential modelling

✓ SSM stacking + normalize + residual connection

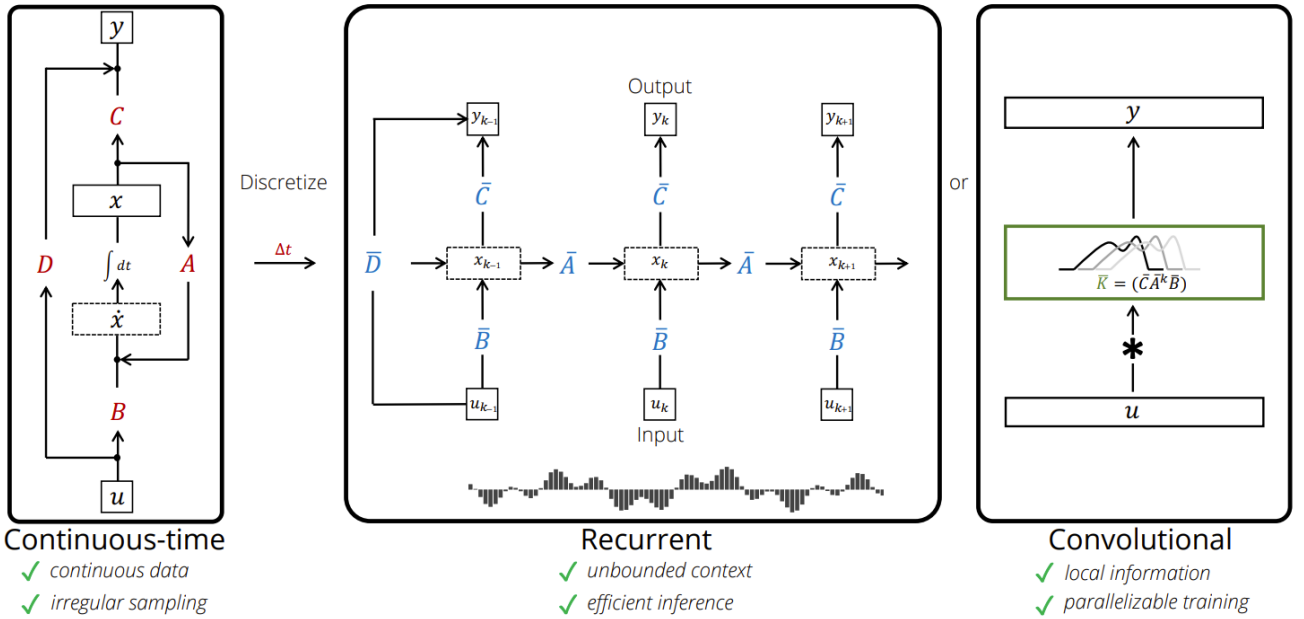
- Even without MLP

✓ 기능적으로 목적이 바뀌는 것은 없음

- Transformer처럼 stacking 을 통해 그 performance를 높이고자 함

✓ Convolution을 통한 fast train,

Recurrence를 통한 efficient inference



Mamba

Towards Mamba!

- LSSL

✓ 파라미터를 어떻게 학습시킬 것인가?

- 이와 관련하여, 특히 A matrix에 집중

A matrix

✓ 왜 A matrix가 중요한가?

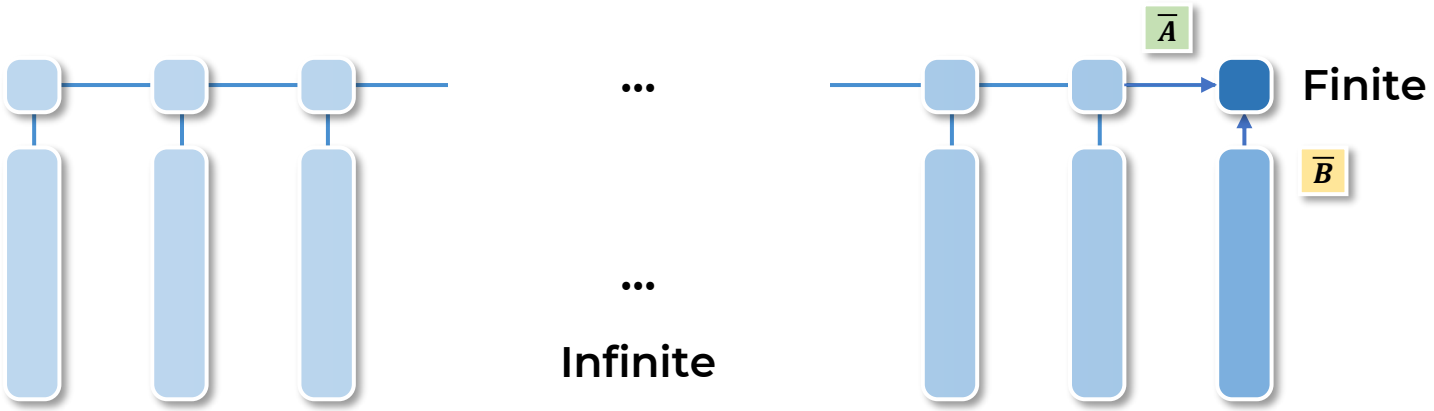
- Memory의 업데이트를 담당
- Gradient의 측면에서 불안정

✓ Random initialized된 A는 optimizing이 어려움

Towards Mamba!

- HiPPO
  - ✓ SSM과는 크게 상관 X
  - ✓ Problem:
    - In learning sequential data, representing cumulative infinite history into finite memory is needed
  - ✓ Objective:
    - Given any function f, memorize it onto some memory state c(t)

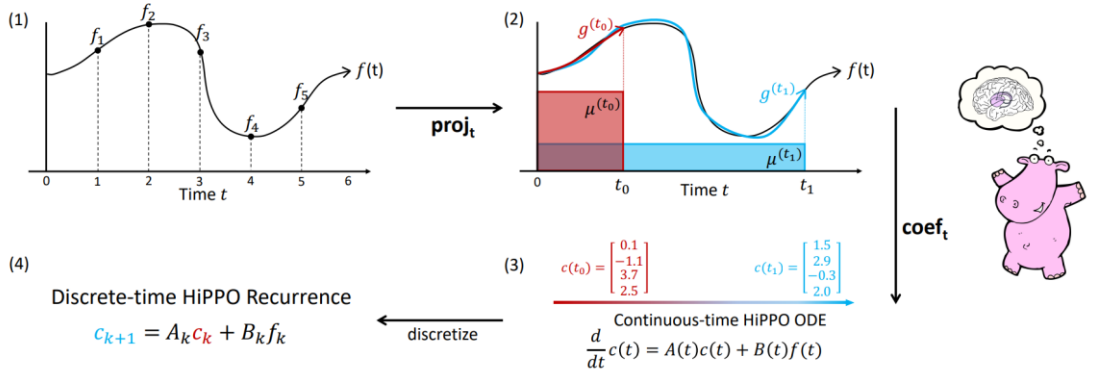
Does it Truly Remember the Whole Function(Input)?



Mamba

Towards Mamba!

- HiPPO
  - ✓ Solution:
    - Approximate it with function g, which is comprised with finite n polynomials
  - ✓ Detail:
    - Given measure w(t) (time-dependant), function g, which is comprised with finite n polynomials, (Ex. Legendre), we can derive c(t) such that, can project the function f (from -inf to t) onto function g, which $c(t)_k$ can be the optimal coefficient of g_k(k_{th} polynomial)



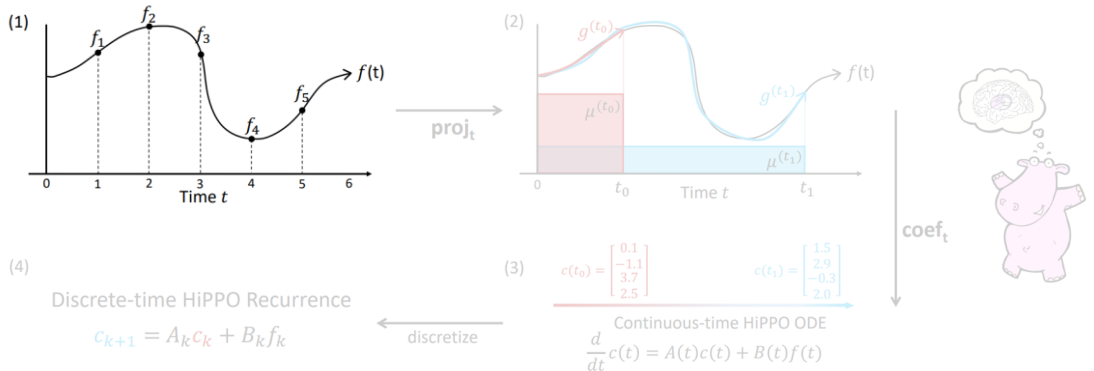
Mamba

Towards Mamba!

- HiPPO

기억해야 하는 Function(Sequence) f가 존재

$f_{\leq t}$를 bounded dimension 내에
compress하여 Memorize해야 함



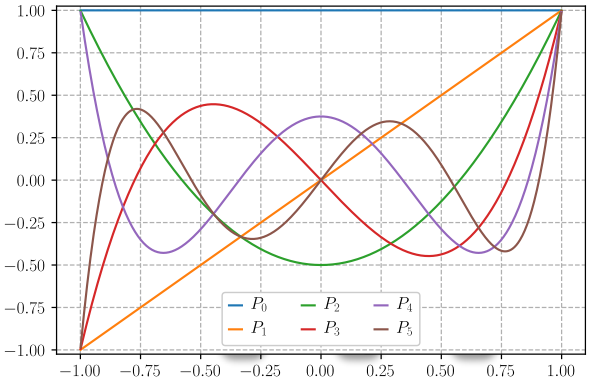
Mamba

Towards Mamba!

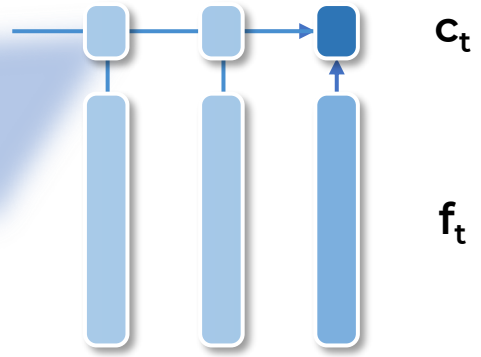
- HiPPO

기억해야 하는 Function(Sequence) f가 존재

f를 controllable function g에 projection



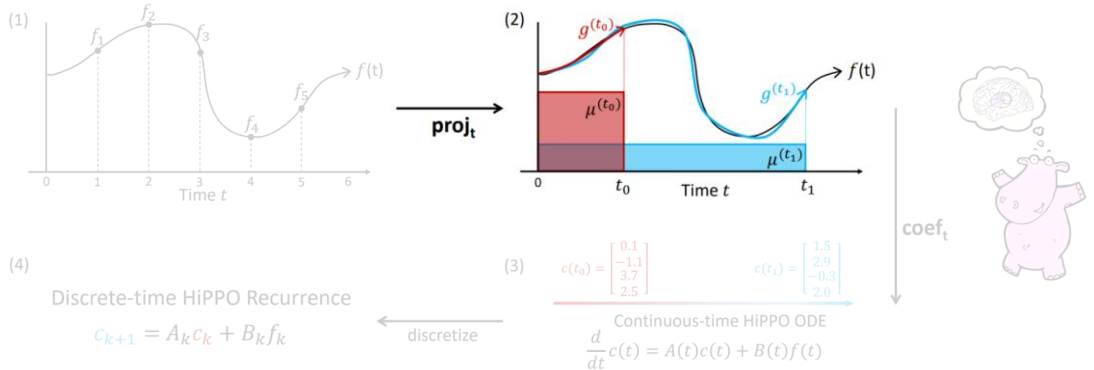
$$c_t = \begin{bmatrix} c_{t1} \\ c_{t2} \\ \cdots \\ c_{tn} \end{bmatrix}$$



f(t): target function
g(t): n orthogonal polynomial basis
μ(t): measure of g(t)
c(t): mapped coefficients for g(t)

C_t should guarantee minimizing

$$\left\| f_{\leq t} - g^{(t)} \right\|_{L2(\mu^{(t)})}$$



Mamba

Towards Mamba!

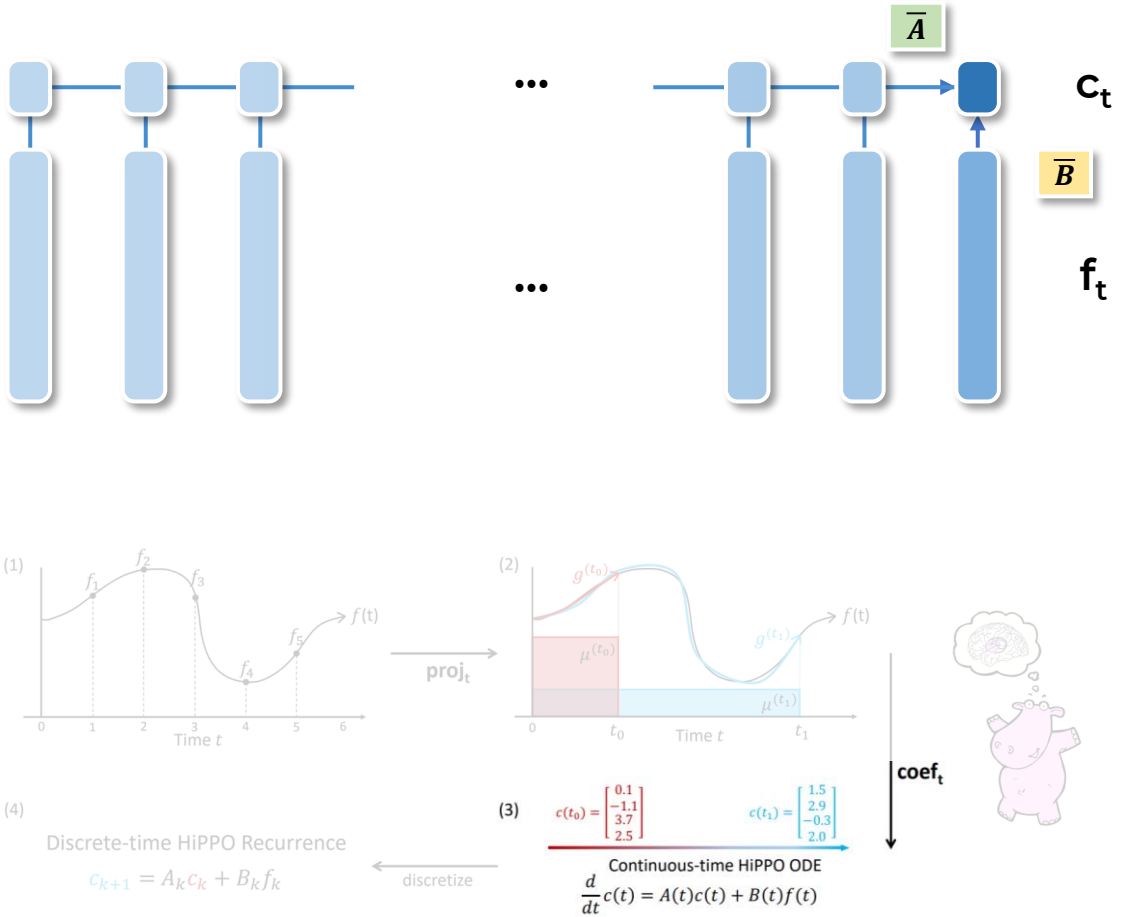
- HiPPO

기억해야 하는 Function(Sequence) f가 존재

f를 controllable function g에 projection

적절한 g와 μ의 선택 → analyzable한 c_t의 update rule

Update rule을 안다는 것 →
For any given timestep t, can project f_t on N-D subspace with c_t at its best analytically



Mamba

Towards Mamba!

- HiPPO

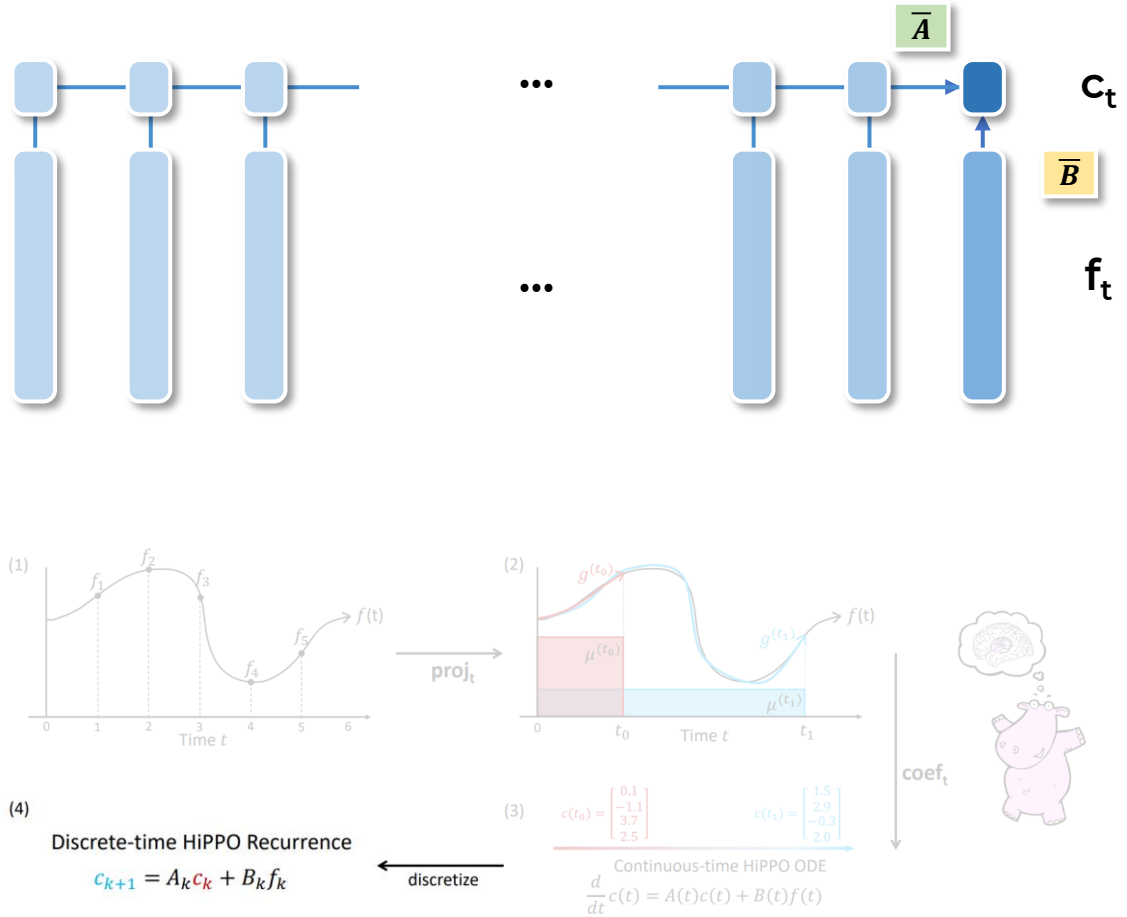
기억해야 하는 Function(Sequence) f가 존재

f를 controllable function g에 projection

적절한 g와 μ의 선택 → analyzable한 c_t의 update rule

Update rule을 안다는 것 →
For any given timestep t, can project f_t on N-D subspace with c_t at its best analytically

Discrete recurrence로 modelling



Mamba

Towards Mamba!

- HiPPO

기억해야 하는 Function(Sequence) f가 존재

f를 controllable function g에 projection

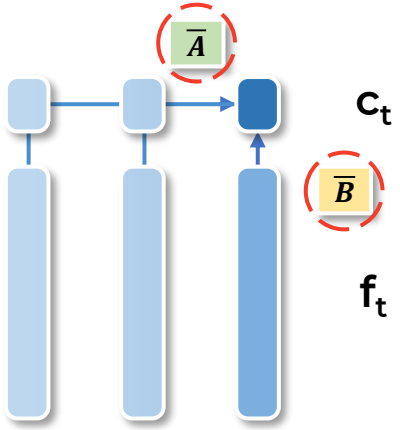
적절한 g와 μ의 선택 → analyzable한 c_t의 update rule

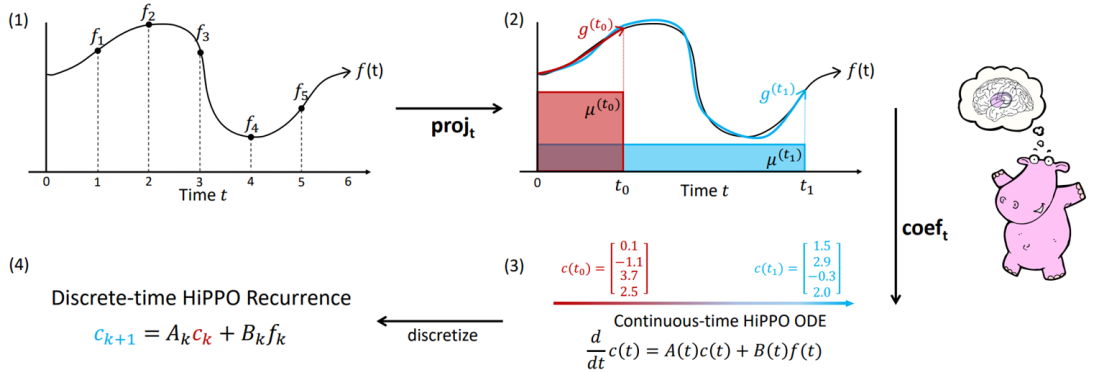
Update rule을 안다는 것 →
For any given timestep t, can project f_t on N-D subspace with c_t at its best analytically

Discrete recurrence로 modelling

적절한 g와 μ 의 선택 →
c(t)의 update rule을 아래와 같은 ODE꼴로
만들어주는 A와 B를 도출할 수 있음

$$A_{nk} = \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & if\ n > k \\ n+1 & if\ n = k \\ 0 & if\ n < k \end{cases} \qquad B_n = (2n+1)^{\frac{1}{2}}$$

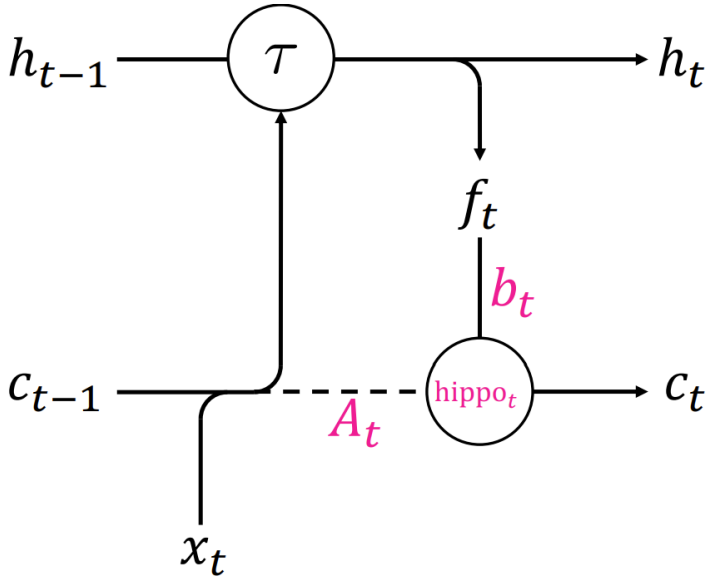




Mamba

Towards Mamba!

- HiPPO



A, B are also derived! Not Trained!

$$A_{nk} = \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases}, \qquad B_n = (2n+1)^{\frac{1}{2}}$$

$$\begin{aligned} h_t \in \mathbb{R}^d &= \tau(h_{t-1}, [c_{t-1}, x_t]) \\ f_t \in \mathbb{R}^1 &= \mathcal{L}_f(h_t) \\ c_t \in \mathbb{R}^N &= \text{hippo}_t(f) \\ &= A_t c_{t-1} + B_t f_t \end{aligned}$$

... So, it trains something looks like this

(which is not that important to us right now)

Mamba

Towards Mamba!

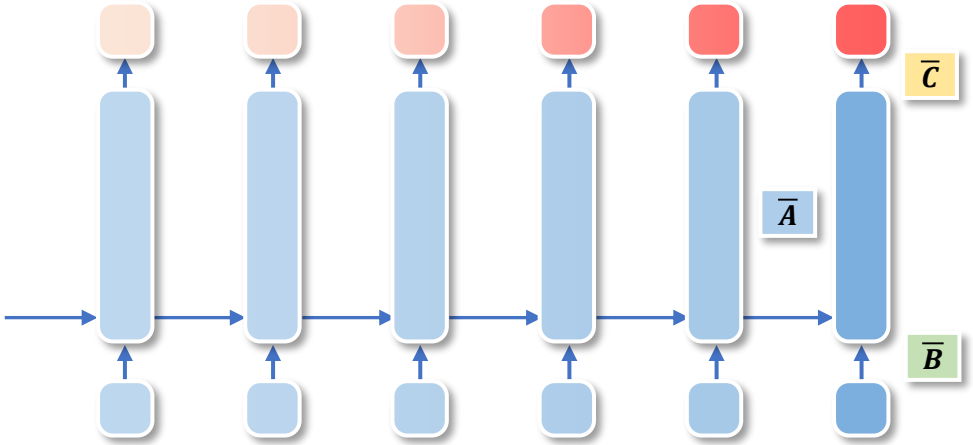
- LSSL
  - ✓ HiPPO 이후로, 거의 모든 deep SSM 모델들은 A를 HiPPO Matrix로 초기화
  - ✓ HiPPO와 달리 A를 trainable하게 바꿈
    - HiPPO로 초기화를 하고, train도 optional하게 할 수 있게 함
    - Relaxes HiPPO to arbitrary measure ω

이제 좀 SSM이라는 걸 train시켜봐도 될까요...?

알아 두셔야 할 게 있습니다
좀 복잡하긴 한데요... 🤔

Towards Mamba!

- LSSL



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

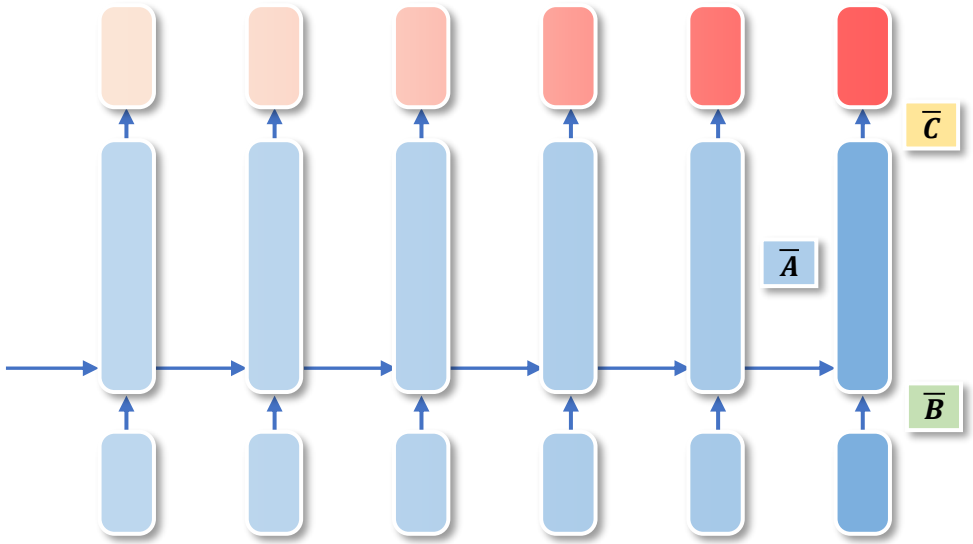
$$\mathbf{y}_t \in \mathbb{R}^1$$
$$\mathbf{x}_t \in \mathbb{R}^n$$
$$\mathbf{u}_t \in \mathbb{R}^1$$

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

Towards Mamba!

- LSSL

🤔 이렇게 되어야 하지 않나요? 🤔



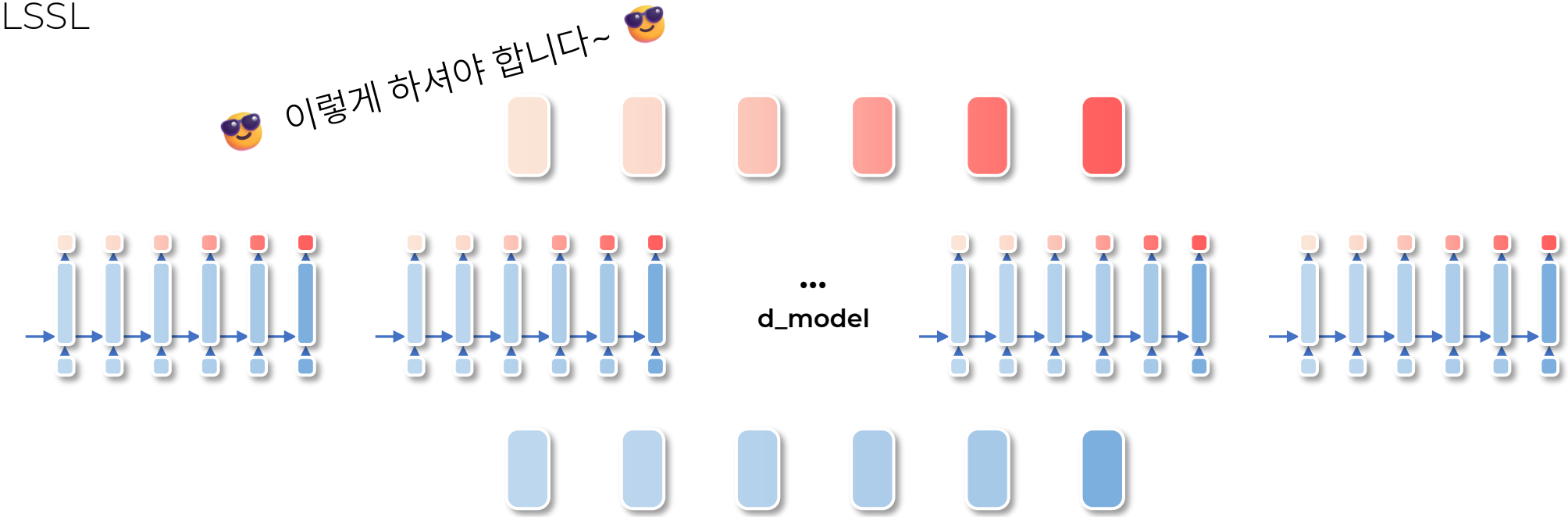
$$\mathbf{x}_t = \bar{\mathbf{A}}\mathbf{x}_{t-1} + \bar{\mathbf{B}}\mathbf{u}_t$$
$$\mathbf{y}_t = \mathbf{C}\mathbf{x}_t$$

$$\mathbf{y}_t \in \mathbb{R}^{d\_model}$$
$$\mathbf{x}_t \in \mathbb{R}^{n}$$
$$\mathbf{u}_t \in \mathbb{R}^{d\_model}$$

$$\bar{\mathbf{A}} \in \mathbb{R}^{n \times n}$$
$$\bar{\mathbf{B}} \in \mathbb{R}^{n \times d\_model}$$
$$\mathbf{C} \in \mathbb{R}^{d\_model \times n}$$

Towards Mamba!

- LSSL



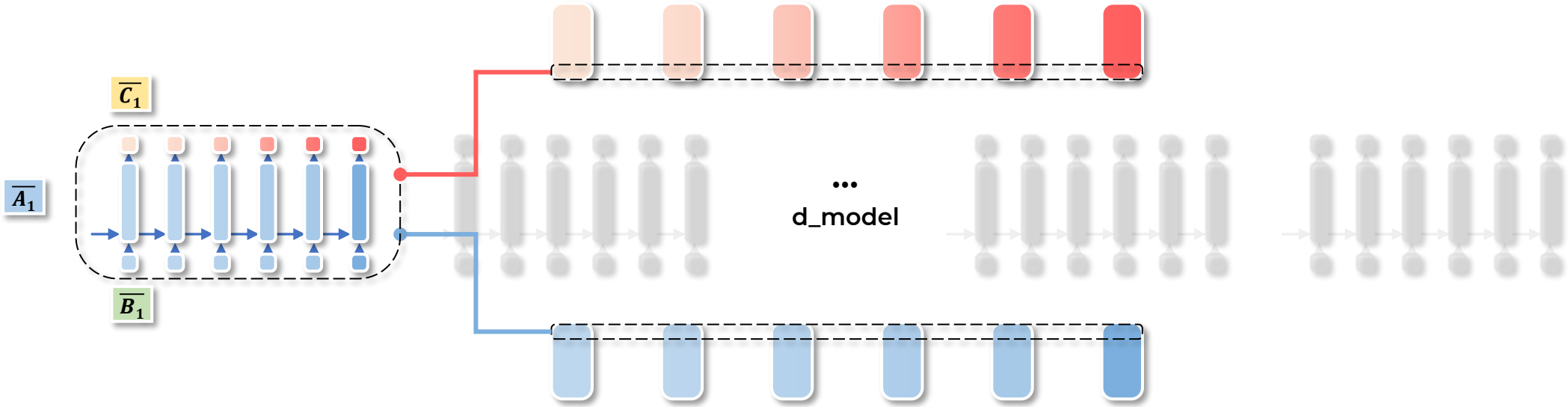
$$\mathbf{x}_t = \bar{\mathbf{A}}\mathbf{x}_{t-1} + \bar{\mathbf{B}}\mathbf{u}_t$$
$$\mathbf{y}_t = \mathbf{C}\mathbf{x}_t$$

$$\mathbf{y}_t \in \mathbb{R}^{d\_model}$$
$$\mathbf{x}_t \in \mathbb{R}^{n}$$
$$\mathbf{u}_t \in \mathbb{R}^{d\_model}$$

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

Towards Mamba!

- LSSL



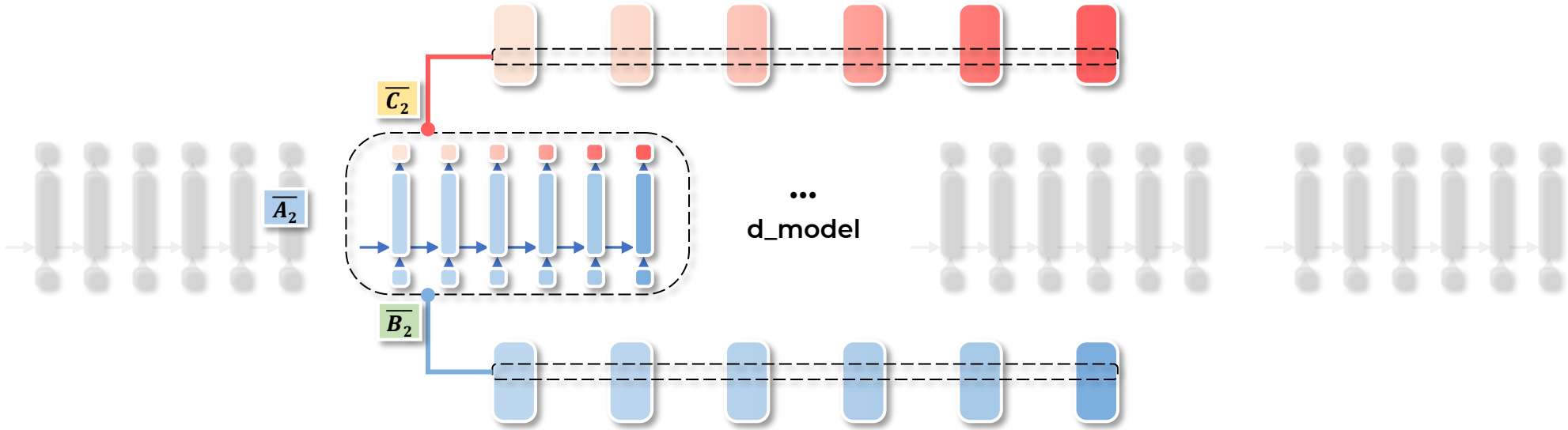
$$\mathbf{x}_t = \bar{\mathbf{A}}\mathbf{x}_{t-1} + \bar{\mathbf{B}}\mathbf{u}_t$$
$$\mathbf{y}_t = \mathbf{C}\mathbf{x}_t$$

$$\mathbf{y}_{t,1} \in \mathbb{R}^1$$
$$\mathbf{x}_{t,1} \in \mathbb{R}^n$$
$$\mathbf{u}_{t,1} \in \mathbb{R}^1$$

$$\bar{\mathbf{A}}_1 \in \mathbb{R}^{n \times n}$$
$$\bar{\mathbf{B}}_1 \in \mathbb{R}^{n \times 1}$$
$$\bar{\mathbf{C}}_1 \in \mathbb{R}^{1 \times n}$$

Towards Mamba!

- LSSL



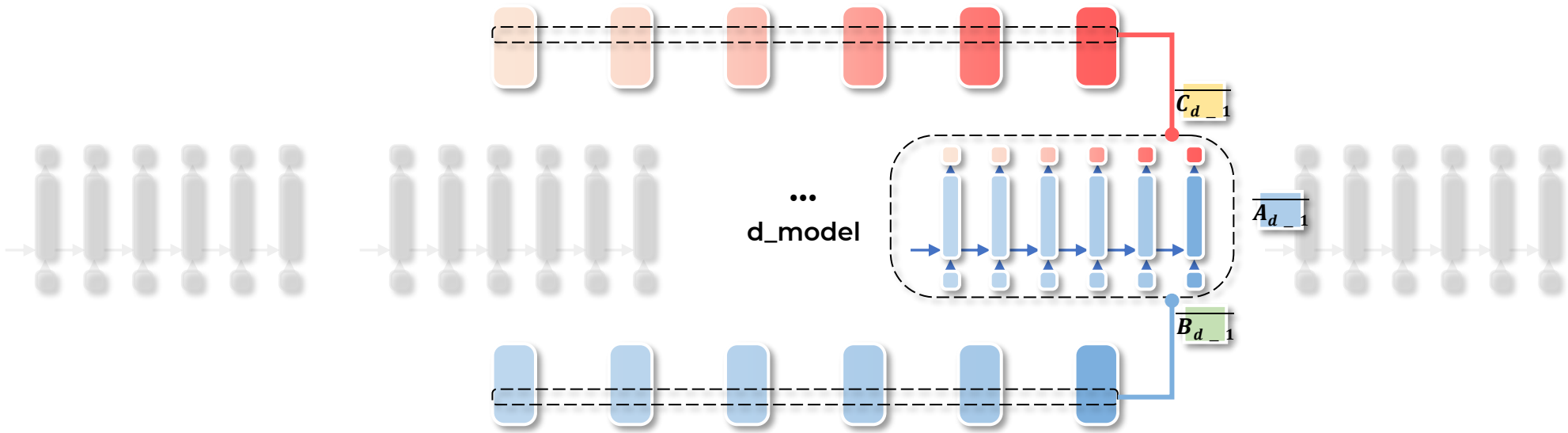
$$\mathbf{x}_t = \bar{\mathbf{A}}\mathbf{x}_{t-1} + \bar{\mathbf{B}}\mathbf{u}_t$$
$$\mathbf{y}_t = \mathbf{C}\mathbf{x}_t$$

$$\mathbf{y}_{t,2} \in \mathbb{R}^1$$
$$\mathbf{x}_{t,2} \in \mathbb{R}^n$$
$$\mathbf{u}_{t,2} \in \mathbb{R}^1$$

$$\bar{\mathbf{A}}_2 \in \mathbb{R}^{n \times n}$$
$$\bar{\mathbf{B}}_2 \in \mathbb{R}^{n \times 1}$$
$$\bar{\mathbf{C}}_2 \in \mathbb{R}^{1 \times n}$$

Towards Mamba!

- LSSL



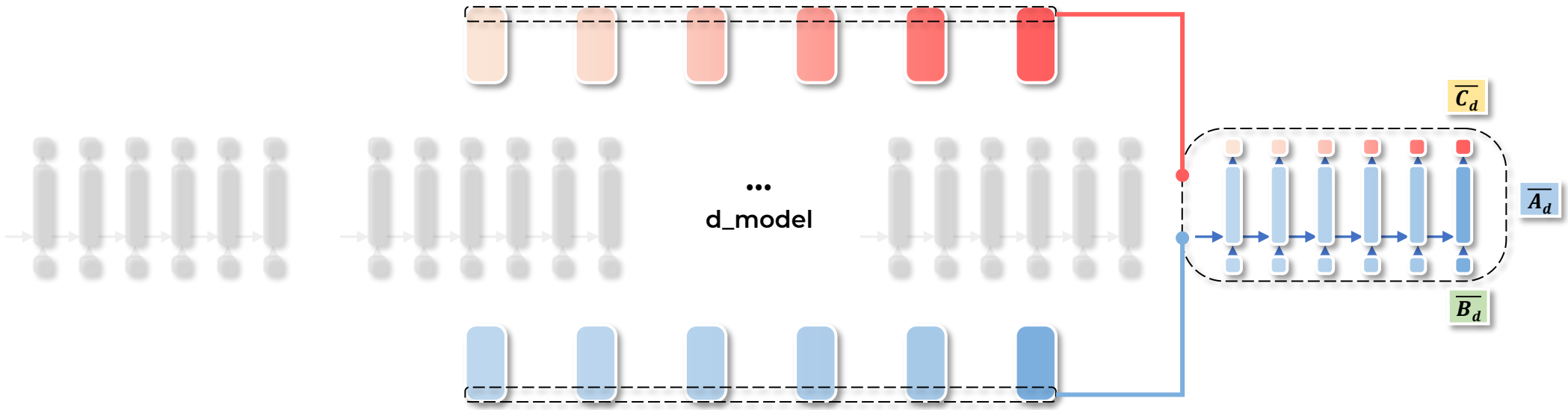
$$\mathbf{x}_t = \bar{\mathbf{A}}\mathbf{x}_{t-1} + \bar{\mathbf{B}}\mathbf{u}_t$$
$$\mathbf{y}_t = \mathbf{C}\mathbf{x}_t$$

$$\mathbf{y}_{t,d-1} \in \mathbb{R}^1$$
$$\mathbf{x}_{t,d-1} \in \mathbb{R}^n$$
$$\mathbf{u}_{t,d-1} \in \mathbb{R}^1$$

$$\bar{\mathbf{A}}_{d\_model-1} \in \mathbb{R}^{n\times n}$$
$$\bar{\mathbf{B}}_{d\_model-1} \in \mathbb{R}^{n\times 1}$$
$$\bar{\mathbf{C}}_{d\_model-1} \in \mathbb{R}^{1\times n}$$

Towards Mamba!

- LSSL



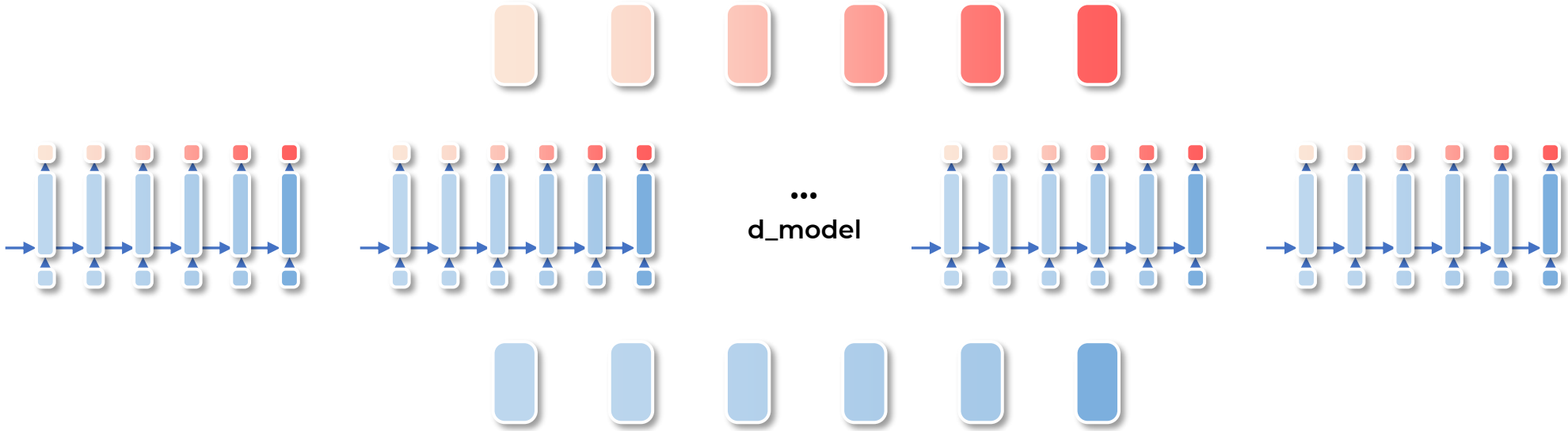
$$\mathbf{x}_t = \bar{\mathbf{A}}\mathbf{x}_{t-1} + \bar{\mathbf{B}}\mathbf{u}_t$$
$$\mathbf{y}_t = \mathbf{C}\mathbf{x}_t$$

$$\mathbf{y}_{t,d} \in \mathbb{R}^1$$
$$\mathbf{x}_{t,d} \in \mathbb{R}^n$$
$$\mathbf{u}_{t,d} \in \mathbb{R}^1$$

$$\bar{\mathbf{A}}_{d\_model} \in \mathbb{R}^{n \times n}$$
$$\bar{\mathbf{B}}_{d\_model} \in \mathbb{R}^{n \times 1}$$
$$\bar{\mathbf{C}}_{d\_model} \in \mathbb{R}^{1 \times n}$$

Towards Mamba!

- LSSL



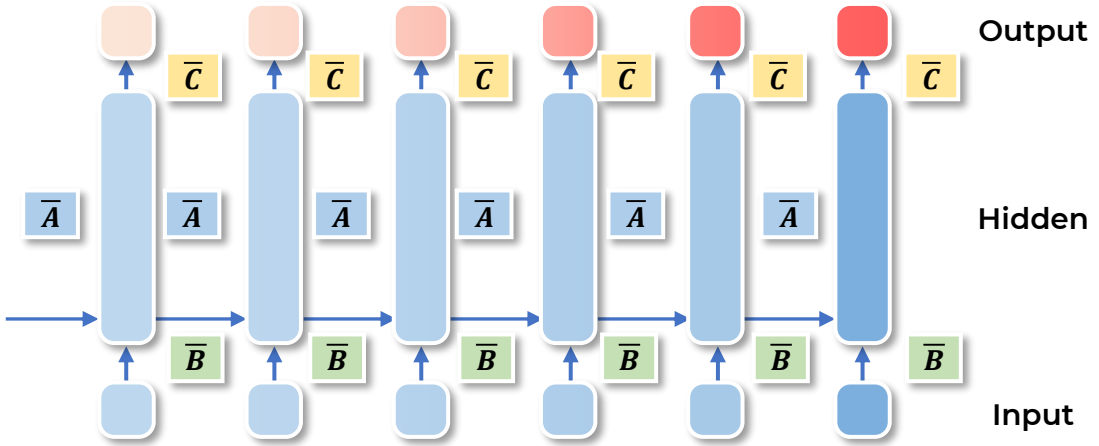
d_model개의 independent한 function에 대한 system을 만드는 것이 핵심!

d_model개의 B vectors, C vectors, A Matrices(If trainable)

Towards Mamba!

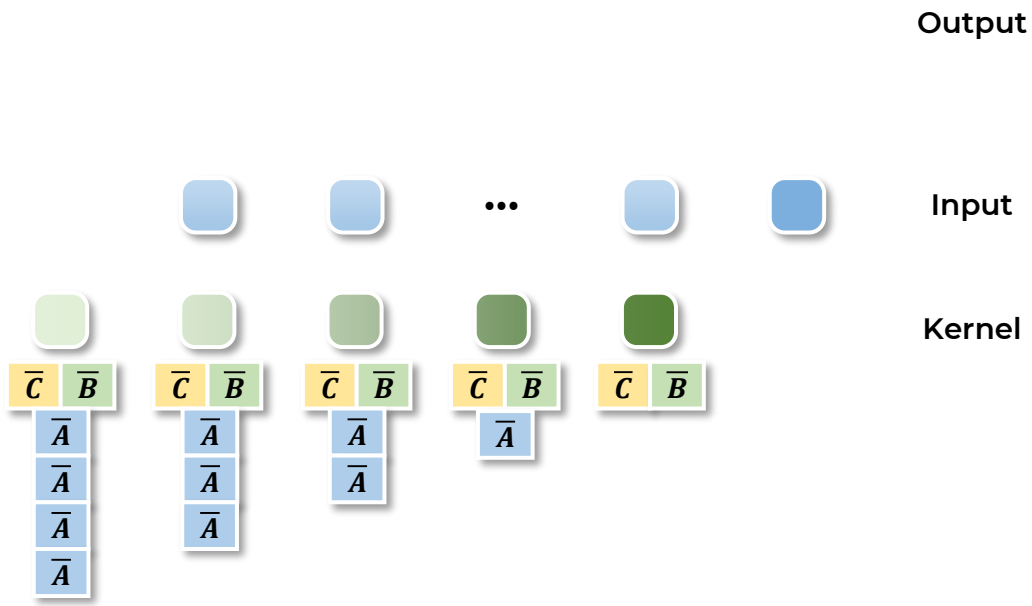
- S4(D)

[Recurrent]



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

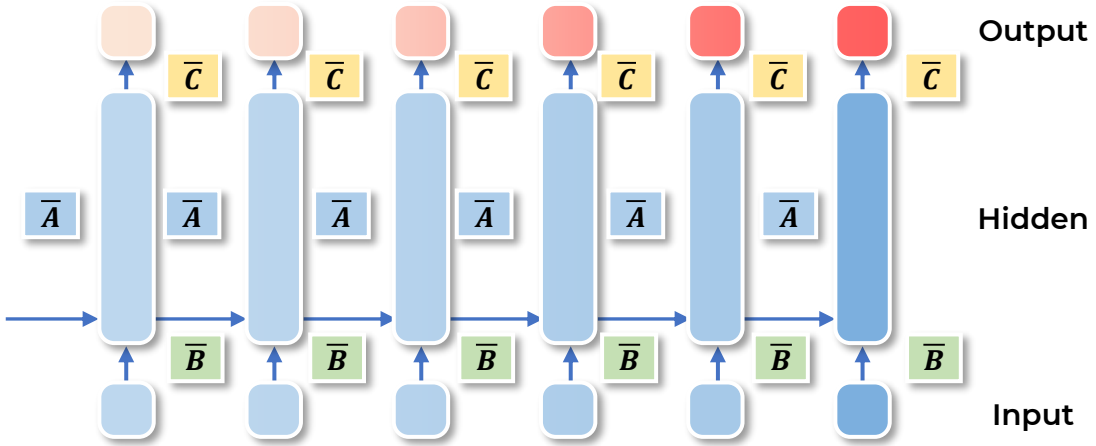
[Convolution]



Towards Mamba!

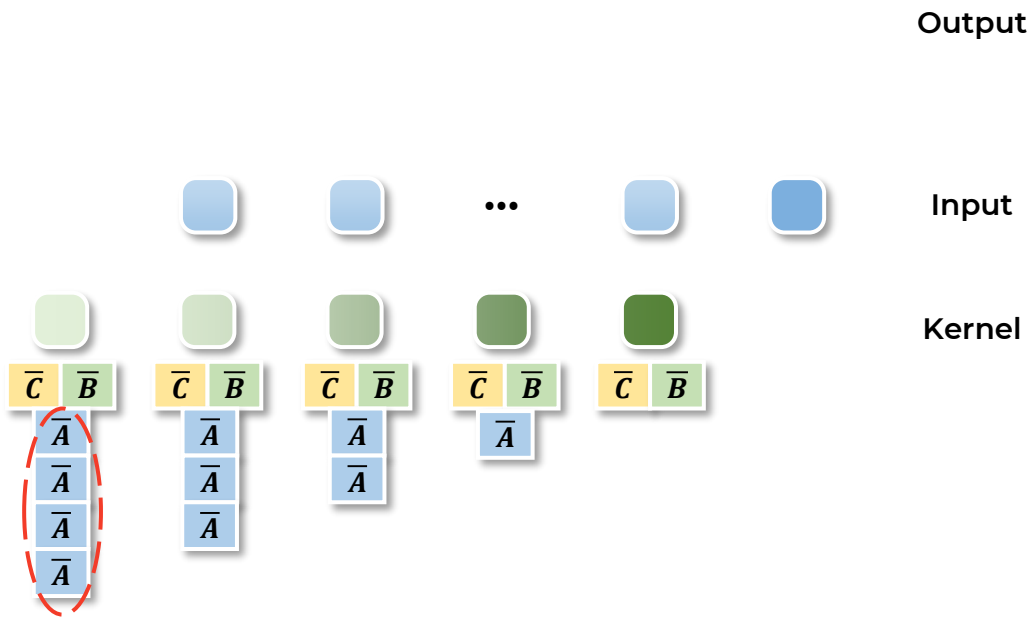
- S4(D)

[Recurrent]



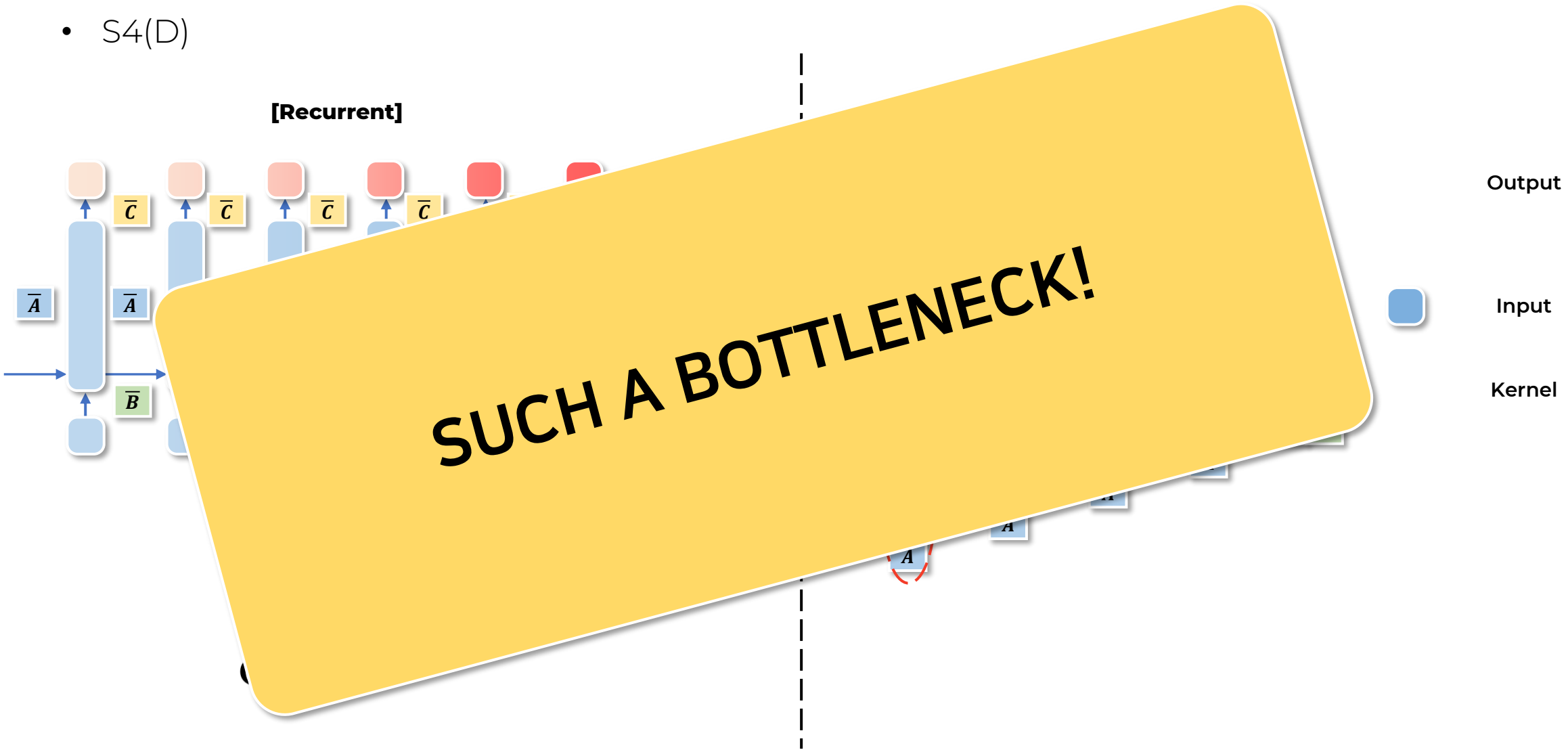
$$\mathbf{A} \in \mathbb{R}^{n \times n}$$
$$\mathbf{B} \in \mathbb{R}^{n \times 1}$$
$$\mathbf{C} \in \mathbb{R}^{1 \times n}$$

[Convolution]



Towards Mamba!

- S4(D)

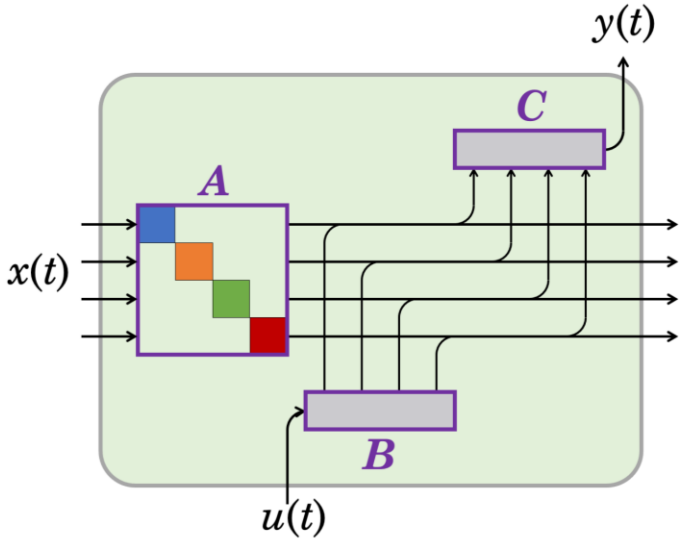


Mamba

Towards Mamba!

- S4(D)
  - ✓ LSSL의 training은 computational 측면에서 사실상 infeasible
  - ✓ 이를 해결하기 위해, reparametrized SSM, Structured State Space Sequence Model(S4) 을 제시
    - S4D에서 사실상 A에 대해 아래와 같이 완전한 Diagonalization을 만들어 냄
  - ✓ Language Modelling에서도 Transformer에 대해 Competitive한 PPL (글쎄...)

A를 안정적으로 대각화하여 연산량을 대폭 낮춤
Deep SSM setting에서도 실질적으로 훈련이 feasible & stable해짐

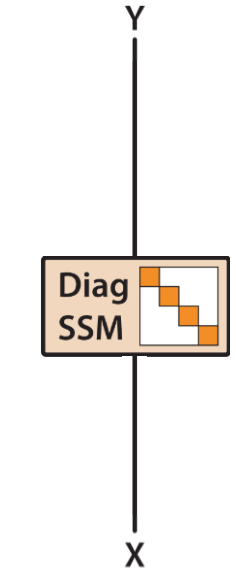


S4D Recurrent View

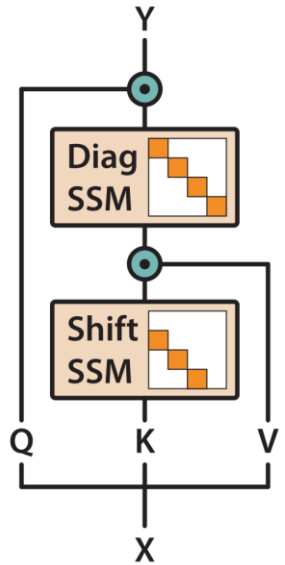
Mamba

Towards Mamba!

- H3
  - ✓ S4 이후, SSM을 더 고도화 하고자 한 논문
    - 특히 Language Modeling에서의 Structured SSM 성능을 올리고자 함
  - ✓ 이를 위해, Language Modeling을 잘 하는 Transformer에 비해, Structured SSM이 무엇을 못하는지 연구
    - Mamba도 후에 이와 비슷한 approach를 취함
  - ✓ 결과적으로, 단순히 SSM을 쌓는 것이 아니라, Transformer와 같이 하나의 block 단위에 좀 더 Complex한 연산이 들어감



Former SSMs



H3 Layer

Mamba

Towards Mamba!

- H3
  - ✓ What does SSM suffer from?

Task	Input	Output
Induction Head	$a\ b\ c\ d\ e \vdash f\ g\ h\ i \ldots x\ y\ z \vdash$	f
Associative Recall	$a\ 2\ c\ 4\ b\ 3\ d\ 1\ a$	2

정해진 Special token 뒤에
어떤 글자가 왔었는지
기억하고 생성해내는 Task

여러 Key-Value 쌍을 기억해서
주어진 Key에 대한 Value를
생성해내는 Task

Task	Random	S4D	Gated State Spaces	H3	Attention
Induction Head	5.0	35.6	6.8	100.0	100.0
Associative Recall	25.0	86.0	78.0	99.8	100.0

어떤 Capability가 없어서 못 하는 걸까?

→ 어떤 Capability를 심어줘야 할까?

Mamba

Towards Mamba!

- H3
  - ✓ What does SSM suffer from?

Task	Input	Output
Induction Head	$a\ b\ c\ d\ e \vdash f\ g\ h\ i \ldots x\ y\ z \vdash$	f
Associative Recall	$a\ 2\ c\ 4\ b\ 3\ d\ 1\ a$	2

정해진 Special token 뒤에
어떤 글자가 왔었는지
기억하고 생성해내는 Task

여러 Key-Value 쌍을 기억해서
주어진 Key에 대한 Value를
생성해내는 Task

Task	Random	S4D	Gated State Spaces	H3	Attention
Induction Head	5.0	35.6	6.8	100.0	100.0
Associative Recall	25.0	86.0	78.0	99.8	100.0

(1)

특정 Input 뒤에 등장한 Input을
기억하는 능력

(2)

지금 시점에 Input된 정보를
지금 시점의 Memory와 비교하는 능력

Mamba

Towards Mamba!

- H3
  - ✓ What does SSM suffer from?

Task	Input	Output
Induction Head	$a\ b\ c\ d\ e \vdash f\ g\ h\ i \ldots x\ y\ z \vdash$	f
Associative Recall	$a\ 2\ c\ 4\ b\ 3\ d\ 1\ a$	2

정해진 Special token 뒤에
어떤 글자가 왔었는지
기억하고 생성해내는 Task

여러 Key-Value 쌍을 기억해서
주어진 Key에 대한 Value를
생성해내는 Task

Task	Random	S4D	Gated State Spaces	H3	Attention
Induction Head	5.0	35.6	6.8	100.0	100.0
Associative Recall	25.0	86.0	78.0	99.8	100.0

(1)

T시점의 Token 정보를 T+m시점까지도
Delay & Convey할 수 있는 능력

(2)

지금 시점에 Input된 정보를
지금 시점의 Memory와 비교하는 능력

Mamba

Towards Mamba!

- H3
  - ✓ 물론, 기존의 SSM도 Global Memory가 있기 때문에, 그런 능력이 있음
    - 실험적으로 잘 못하는 것을 확인
    - 그런 능력을 추가하여 Transformer와의 간극을 줄이고자 하는 내용으로 이해
  - ✓ 특히, (1)을 Input에 대해 Sequence 방향으로의 Conv1D Layer를 추가하는 것으로 쉽게 해결
  - ✓ 중요한 건, SSM이 Language Modelling을 하는 데에 중요한 특정 능력이 부족했고,
    이를 해결하는 Trick중 하나로 Conv1D를 Layer 초반에 넣어줬다는 점

$$\mathbf{Q} \odot \mathrm{SSM}_{\mathrm{diag}}\left(\mathrm{SSM}_{\mathrm{shift}}(\mathbf{K}) \odot \mathbf{V}\right)$$

(1) **T시점의 Token 정보를 T+m시점까지도 Delay & Convey할 수 있는 능력**

(2) **지금 시점에 Input된 정보를 지금 시점의 Memory와 비교하는 능력**

Welcome, Mamba!

Welcome, Mamba!

Welcome, Mamba!

- In Brief; Again

✓ Transformer의 High Performance와 Domain-agnostic한 Robustness는 유지하면서 Sub-quadratic한 architecture에 대한 연구가 지속되어왔지만, 만족할만한 성능은 내지 못했음

SSSSSS...

✓ Mamba는 그 갈래 중 하나인 deep structured SSM의 철학을 이어받는 연구

- 이쪽 계열의 기존 장점인 Fast Inference는 물론,
- High-performance를 달성
- 특히, Language Modelling에도 정말 Transformer와 비견될 정도의 성능을 보여 줌

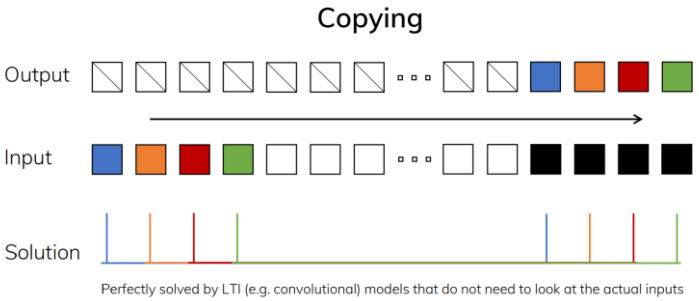
✓ 이를 위해,

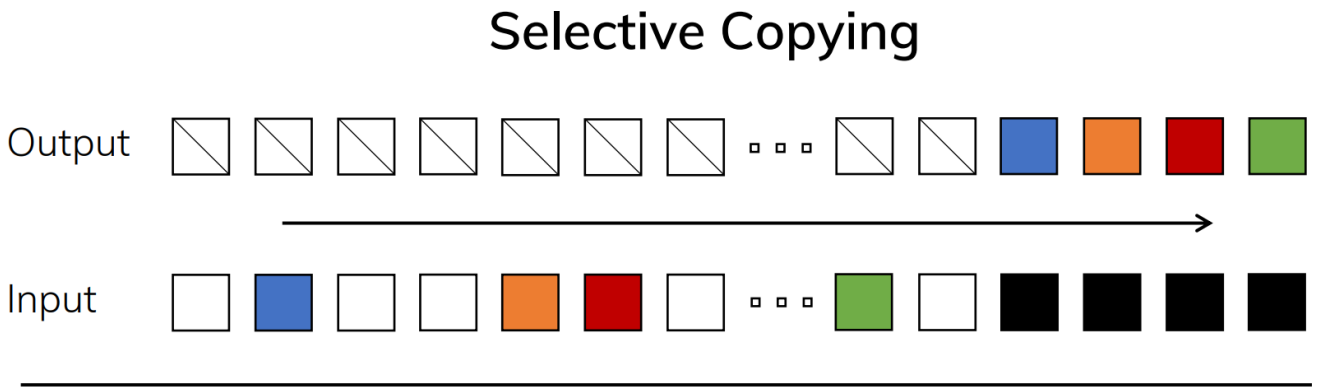
- SSM의 성능 문제는 Selectivity를 도입하여 해결
- 이로 인한 연산 속도의 문제는 Hardware-aware Parallel Scan으로 해결

Mamba

Welcome, Mamba!

- What's the Matter with SSMs
  - ✓ SSMs는 RNN과 CNN의 Interchangeable한 Combination
    - 그로서 LRA, Audio, Vision 등, 특히 Continuous한 Domain에서는 Dominant한 성능을 보여줌
    - 하지만, Text와 같은 info-dense하고 discrete한 분야에서는 약함
  - ✓ 이와 관련해서, H3와 비슷한 Approach를 취함
    - 앞에서 봤던 Inductive Heads에 더하여, 새로운 Synthetic task를 하나 더 고안
    - Selective Copying

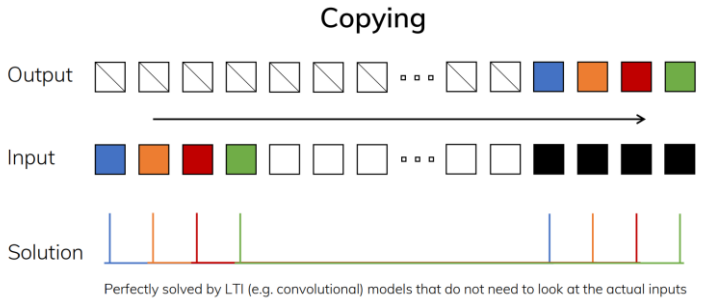




Mamba

Welcome, Mamba!

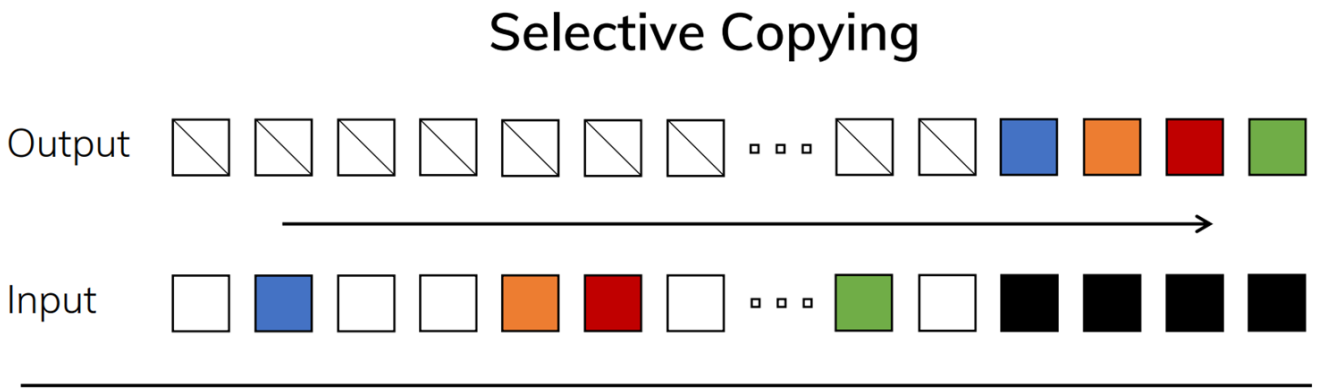
- What's the Matter with SSMs
  - ✓ Sequence가 주어졌을 때, 그대로 Convey하는 것은 기존의 SSM도 할 수 있음
    - 아래와 같은 Solution Kernel을 학습
    - Input에 따라 달라지지 않는 System으로도 할 수 있음



Mamba

Welcome, Mamba!

- What's the Matter with SSMs
  - ✓ 하지만, Input들 중 선택적으로 copying하는 task도 SSM이 잘 할 수 있을까?
    - Hidden Memory h_t에 x_t를 선택적으로 update할 수 있어야 함
    - 특정 token들만 Hidden Memory에 싣는 것이 가능해야 함
  - ✓ S4의 구조로 이가 가능할까?



Mamba

Welcome, Mamba!

- What's the Matter with SSMs

Continuous

$$h'(t) = Ah(t) + Bx(t)$$
$$y(t) = Ch(t)$$

ZOH Discretization

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

Discrete

$$h_t = \overline{A}h_{t-1} + \overline{B}x_t$$
$$y_t = \overline{C}h_t$$

Convolution

$$\overline{K} = (C\overline{B}, C\overline{AB}, ..., C\overline{A^kB}, ...)$$
$$y = x * \overline{K}$$

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

Mamba

Welcome, Mamba!

- What's the Matter with SSMs

$$h_t = \overline{A}h_{t-1} + \overline{B}x_t$$
$$y_t = \overline{C}h_t$$

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

$$x_t \in \mathbb{R}^{\mathrm{D}}$$
$$h_t \in \mathbb{R}^{\mathrm{N} \times \mathrm{D}}$$
$$y_t \in \mathbb{R}^{\mathrm{D}}$$

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: $\Delta : (\mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter})$ $\tau_\Delta = \text{softplus}$
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{D}, \mathtt{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
7: **return** y

Mamba

Welcome, Mamba!

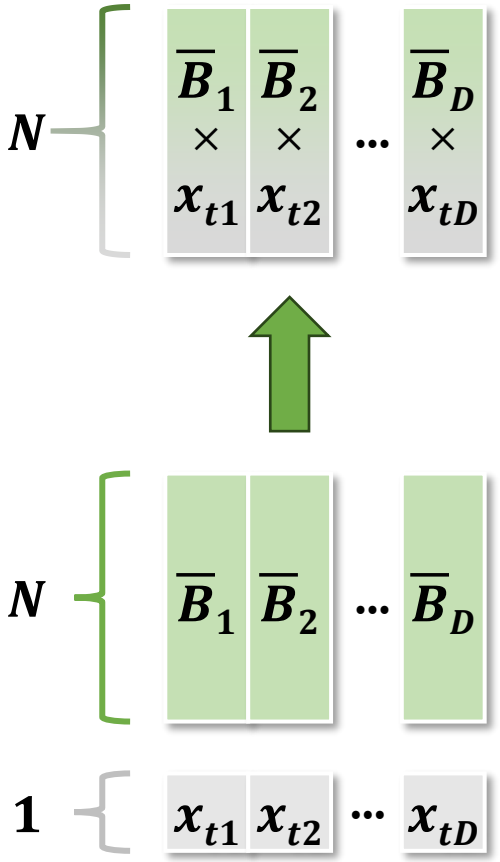
- What's the Matter with SSMs

$$h_t = \overline{A}h_{t-1} + \overline{B}x_t$$
$$y_t = \overline{C}h_t$$

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

$$x_t \in \mathbb{R}^{\mathrm{D}}$$
$$h_t \in \mathbb{R}^{\mathrm{N} \times \mathrm{D}}$$
$$y_t \in \mathbb{R}^{\mathrm{D}}$$

B를 통해 각 D개의 Scalar가 N차원의 Hidden Space에 Projection됨



x는 D개의 Scalar를 가짐

Mamba

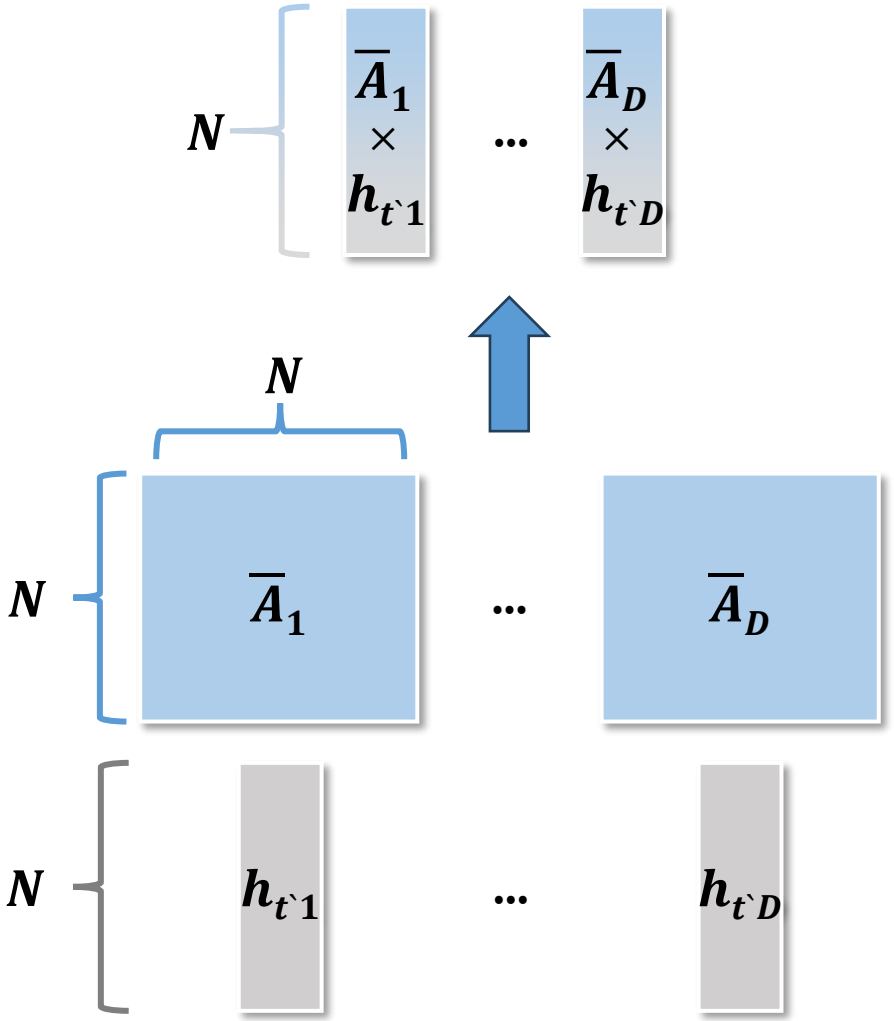
Welcome, Mamba!

- What's the Matter with SSMs

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

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

$$x_t \in \mathbb{R}^{\mathrm{D}}$$
$$h_t \in \mathbb{R}^{\mathrm{N} \times \mathrm{D}}$$
$$y_t \in \mathbb{R}^{\mathrm{D}}$$



Mamba

Welcome, Mamba!

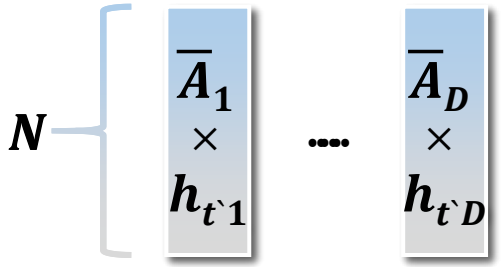
- What's the Matter with SSMs

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

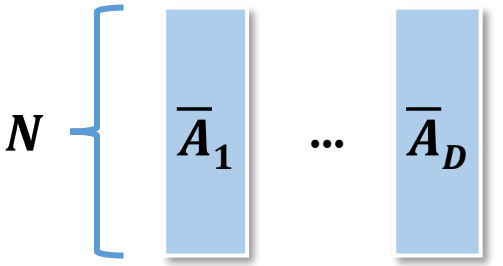
$$\mathbf{A} \in \mathbb{R}^{n \times n \times d\_model}$$
$$\mathbf{B} \in \mathbb{R}^{n \times 1 \times d\_model}$$
$$\mathbf{C} \in \mathbb{R}^{1 \times n \times d\_model}$$

$$x_t \in \mathbb{R}^{\mathrm{D}}$$
$$h_t \in \mathbb{R}^{\mathrm{N} \times \mathrm{D}}$$
$$y_t \in \mathbb{R}^{\mathrm{D}}$$

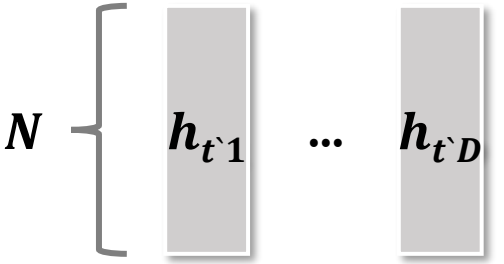
D_i번째 hidden state는
D_i번째 A에 의해
Linear transformed



N x N 크기의 Diagonal Matrix가
D개 있음



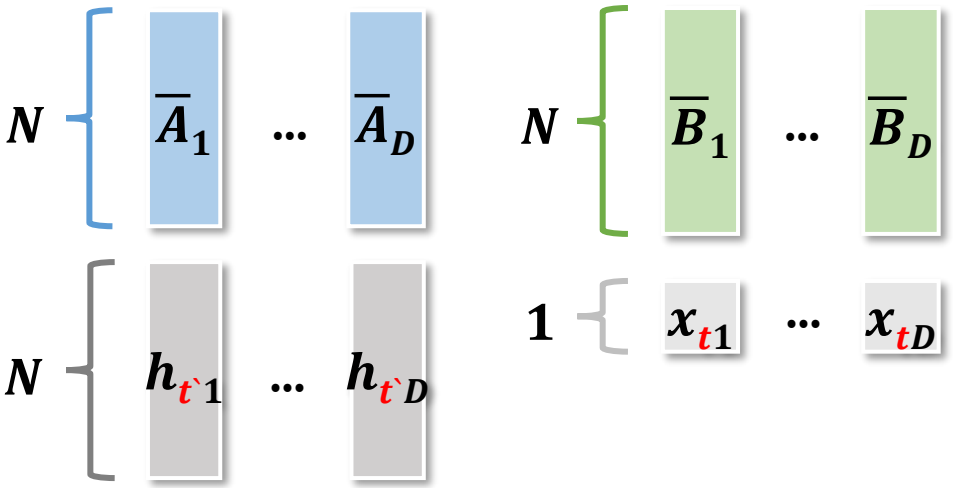
h_{t-1}은 N차원의 hidden state를
각각의 D마다 보유



Mamba

Welcome, Mamba!

- What's the Matter with SSMs



Algorithm 1 SSM (S4)

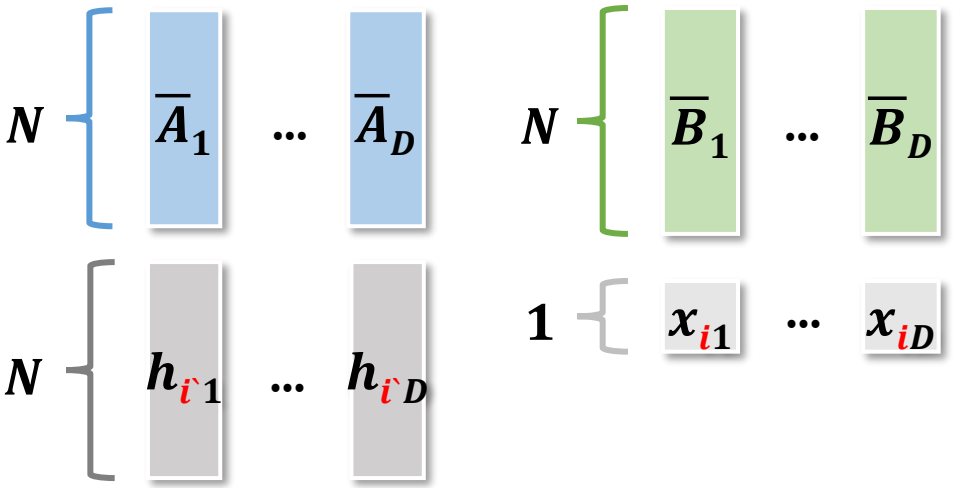
Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$

$\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
$\boldsymbol{C} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
$\Delta : (\mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter})$ $\tau_\Delta = \text{softplus}$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{D}, \mathtt{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
return y

Mamba

Welcome, Mamba!

- What's the Matter with SSMs



Parameters are Time-Invariant!
X값이 무엇이 들어오든 똑같은 Transition을 거친 뒤
Hidden state에 싣게 됨

Algorithm 1 SSM (S4)

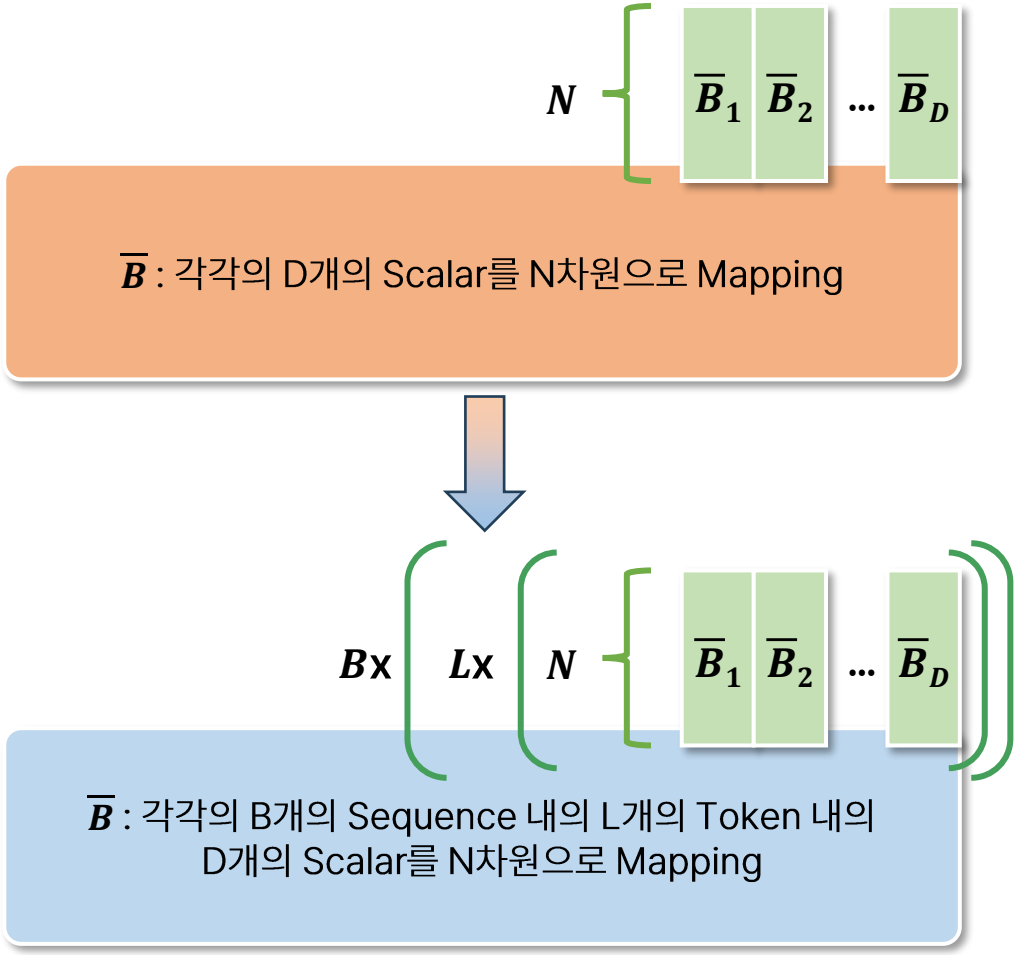
Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$

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: $\Delta : (\mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter})$ $\tau_\Delta = \text{softplus}$
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{D}, \mathtt{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
6: $y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-invariant: recurrence or convolution
7: **return** y

Mamba

Welcome, Mamba!

- Improving SSMs with Selection



Algorithm 2 SSM + Selection (S6)

Input: x : (B, L, D)
Output: y : (B, L, D)
1: $\boldsymbol{A}$: (D, N) ← Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B}$: (B, L, N) ← $s_B(x)$
3: $\boldsymbol{C}$: (B, L, N) ← $s_C(x)$
4: Δ : (B, L, D) ← τ_Δ(Parameter+$s_\Delta(x)$) τ_Δ = softplus
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}$: (B, L, D, N) ← 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

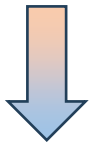
Mamba

Welcome, Mamba!

- Improving SSMs with Selection

2: $\boldsymbol{B}$: (D, N) ← Parameter
3: $\boldsymbol{C}$: (D, N) ← Parameter
4: Δ : (D) ← τ_Δ(Parameter)

$\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{\Delta}$: 그 자체로 Learnt Parameters
Ex. x를 변화시키는 B를 학습



$s_B(x) = \text{Linear}_N(x)$
$s_C(x) = \text{Linear}_N(x)$
$s_\Delta(x) = \text{Broadcast}_D(\text{Linear}_1(x))$

$\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{\Delta}$: x를 Input으로 하는 함수의 결과물
Ex) x를 변화시키는 B를,
x에 따라 각각 만들어낼 수 있는 $S_B(x)$를 학습

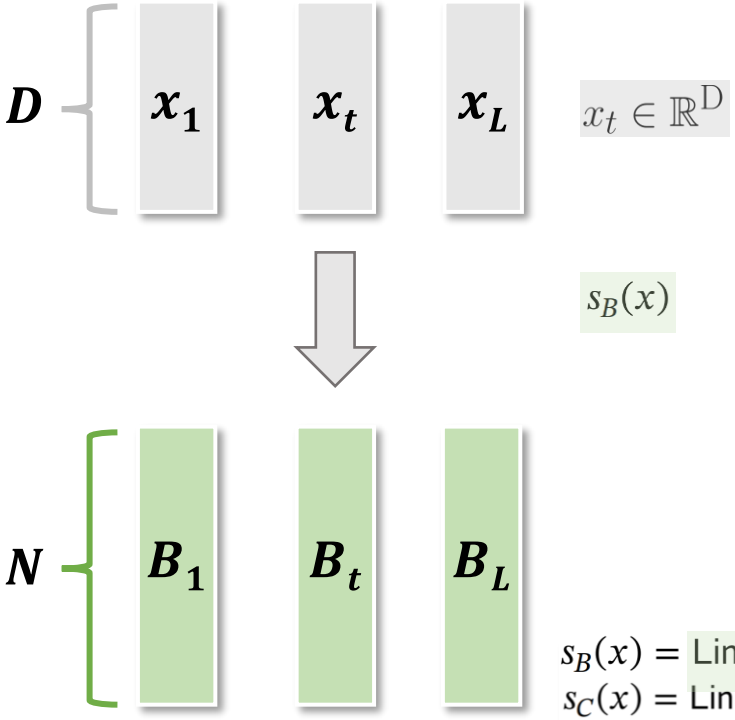
Algorithm 2 SSM + Selection (S6)

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

Mamba

Welcome, Mamba!

- Improving SSMs with Selection



$S_B(x), S_C(x)$: x의 D dimension을 N dimension으로 Projection Input으로 하는 함수의 결과물

Algorithm 2 SSM + Selection (S6)

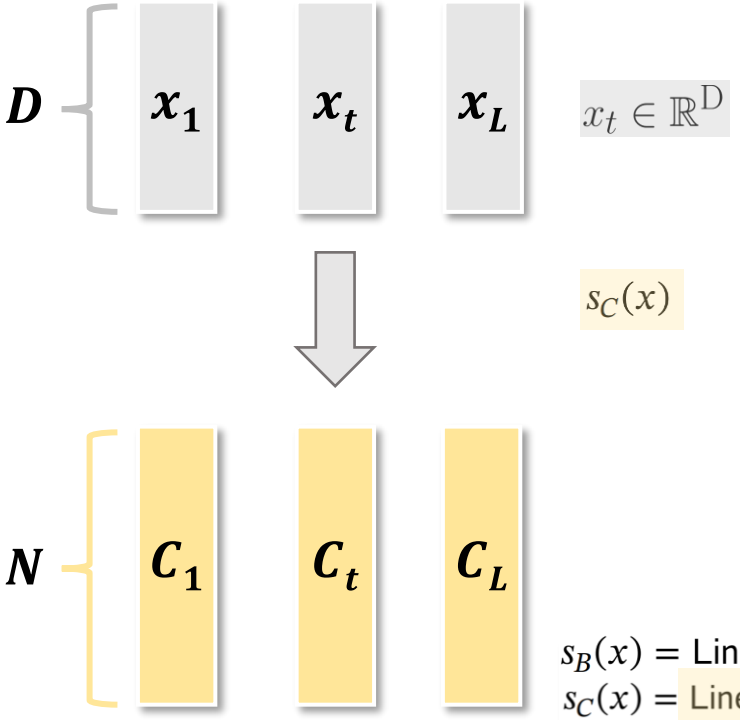
Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$

$\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_B(x)$
$\boldsymbol{C} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_C(x)$
$\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$ $\tau_\Delta = \text{softplus}$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
return y

Mamba

Welcome, Mamba!

- Improving SSMs with Selection



$S_B(x), S_C(x)$: x의 D dimension을 N dimension으로 Projection Input으로 하는 함수의 결과물

Algorithm 2 SSM + Selection (S6)

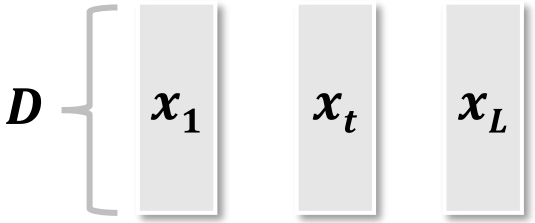
Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$

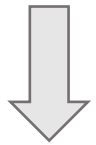
$\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_B(x)$
$\boldsymbol{C} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_C(x)$
$\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$ $\tau_\Delta = \text{softplus}$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
return y

Mamba

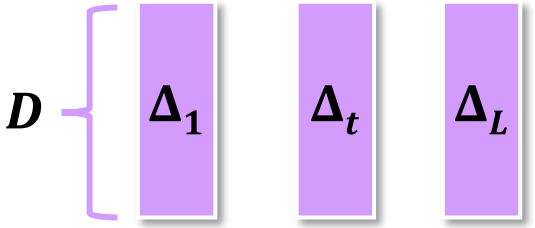
Welcome, Mamba!

- Improving SSMs with Selection





$\tau_\Delta(\text{Parameter}+s_\Delta(x))$



$s_B(x) = \text{Linear}_N(x)$
$s_C(x) = \text{Linear}_N(x)$
$s_\Delta(x) = \text{Broadcast}_D(\text{Linear}_1(x))$

$s_\Delta(x)$: x의 D dimension을 1 dimension으로 Projection한 후 D dimension으로 Broadcast

Algorithm 2 SSM + Selection (S6)

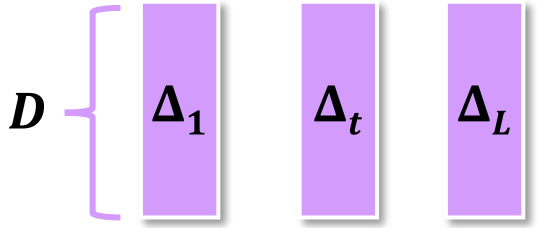
Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$

$\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_B(x)$
$\boldsymbol{C} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_C(x)$
$\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter}+s_\Delta(x))$ $\tau_\Delta = \text{softplus}$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
return y

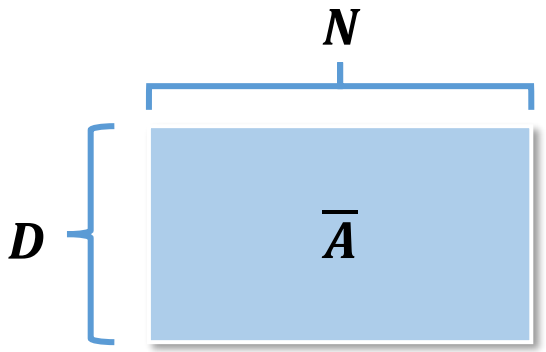
Mamba

Welcome, Mamba!

- Improving SSMs with Selection



$\Delta_t \in \mathbb{R}^{+\mathbb{D}}$



$$\overline{A} = \exp(\Delta A)$$

$\overline{A}$:기존에는 모든 문장(Across B), 모든 토큰의(Across L)
각 Input Scalar(Each D)에 대한 N projection

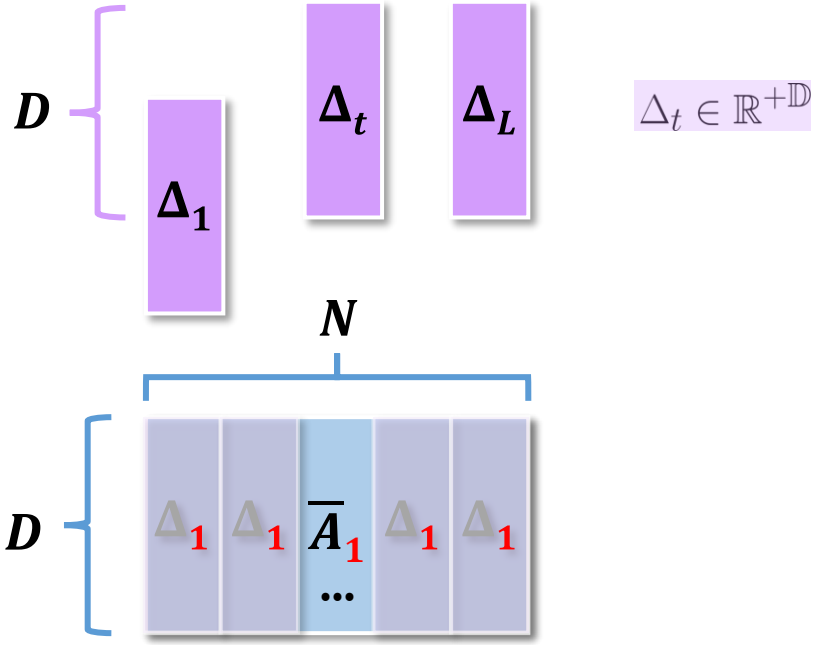
Algorithm 2 SSM + Selection (S6)

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: $\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$ $\tau_\Delta = \text{softplus}$
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

Mamba

Welcome, Mamba!

- Improving SSMs with Selection



$\Delta_t \in \mathbb{R}^{+\mathbb{D}}$

$$\overline{A} = \exp(\Delta A)$$

$\overline{A}$:각 문장(Each B), 각 토큰의(Each L)
각 Input Scalar(Each D)에 대한 NxN Transformation

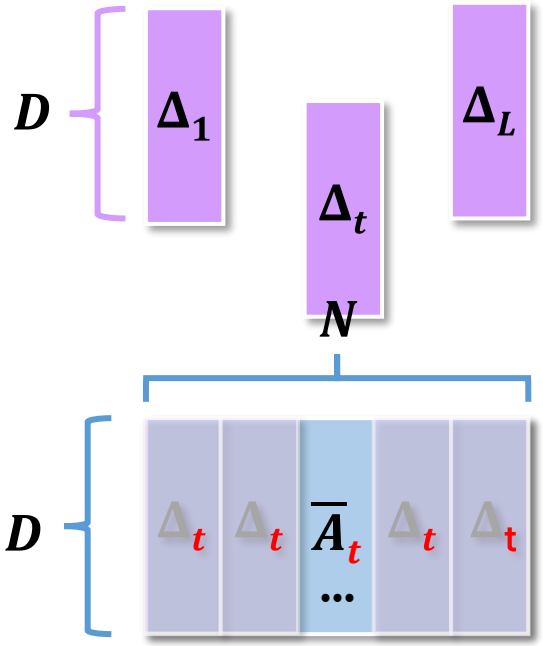
Algorithm 2 SSM + Selection (S6)

Input: x : (B, L, D)
Output: y : (B, L, D)
1: $\boldsymbol{A}$: (D, N) ← Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B}$: (B, L, N) ← $s_B(x)$
3: $\boldsymbol{C}$: (B, L, N) ← $s_C(x)$
4: Δ : (B, L, D) ← τ_Δ(Parameter+$s_\Delta(x)$) τ_Δ = softplus
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}$: (B, L, D, N) ← 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

Mamba

Welcome, Mamba!

- Improving SSMs with Selection



$\Delta_t \in \mathbb{R}^{+\mathbb{D}}$

$$\overline{A} = \exp(\Delta A)$$

$\overline{A}$:각 문장(Each B), 각 토큰의(Each L)
각 Input Scalar(Each D)에 대한 NxN Transformation

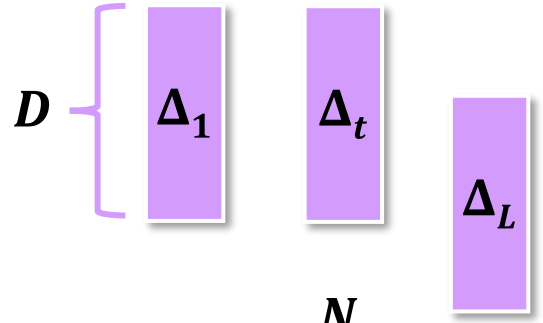
Algorithm 2 SSM + Selection (S6)

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

Mamba

Welcome, Mamba!

- Improving SSMs with Selection



$\Delta_t \in \mathbb{R}^{+\mathbb{D}}$

$$\overline{A} = \exp(\Delta A)$$

$\overline{A}$:각 문장(Each B), 각 토큰의(Each L)
각 Input Scalar(Each D)에 대한 NxN Transformation

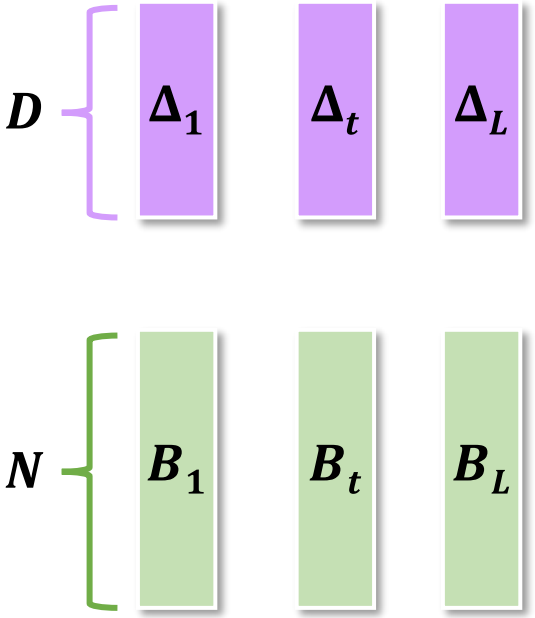
Algorithm 2 SSM + Selection (S6)

Input: x : (B, L, D)
Output: y : (B, L, D)
1: $\boldsymbol{A}$: (D, N) ← Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B}$: (B, L, N) ← $s_B(x)$
3: $\boldsymbol{C}$: (B, L, N) ← $s_C(x)$
4: Δ : (B, L, D) ← τ_Δ(Parameter+$s_\Delta(x)$) τ_Δ = softplus
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}$: (B, L, D, N) ← 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

Mamba

Welcome, Mamba!

- Improving SSMs with Selection



$\Delta_t \in \mathbb{R}^{+\mathbb{D}}$

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

$\overline{\boldsymbol{B}}$: 각 문장(Each B), 각 토큰의(Each L)
각 Input Scalar(Each D)에 대한 N projection

Algorithm 2 SSM + Selection (S6)

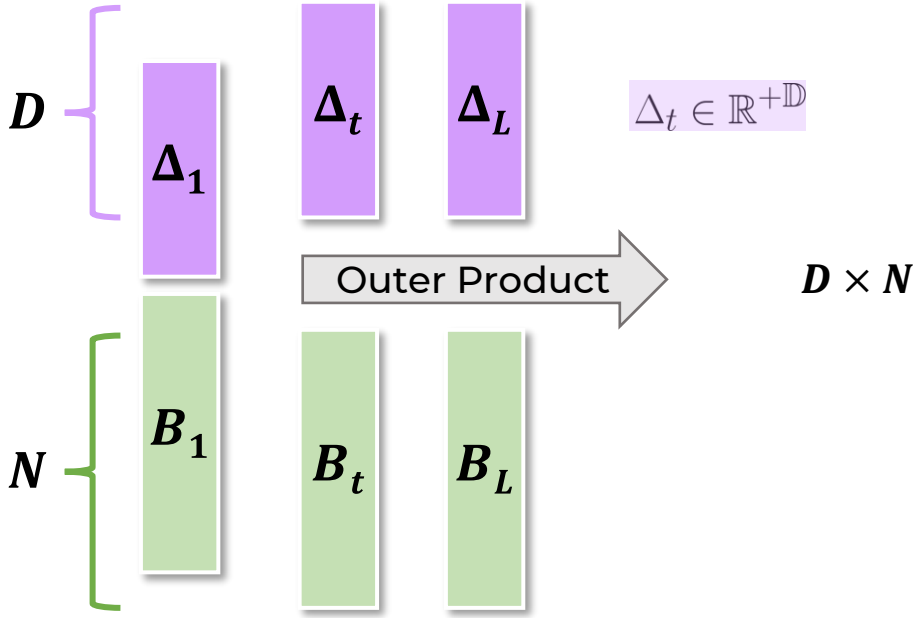
Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$

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: $\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$ $\tau_\Delta = \text{softplus}$
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

Mamba

Welcome, Mamba!

- Improving SSMs with Selection



$\Delta_t \in \mathbb{R}^{+\mathbb{D}}$

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

$\overline{B}$: 각 문장(Each B), 각 토큰의(Each L)
각 Input Scalar(Each D)에 대한 N projection

Algorithm 2 SSM + Selection (S6)

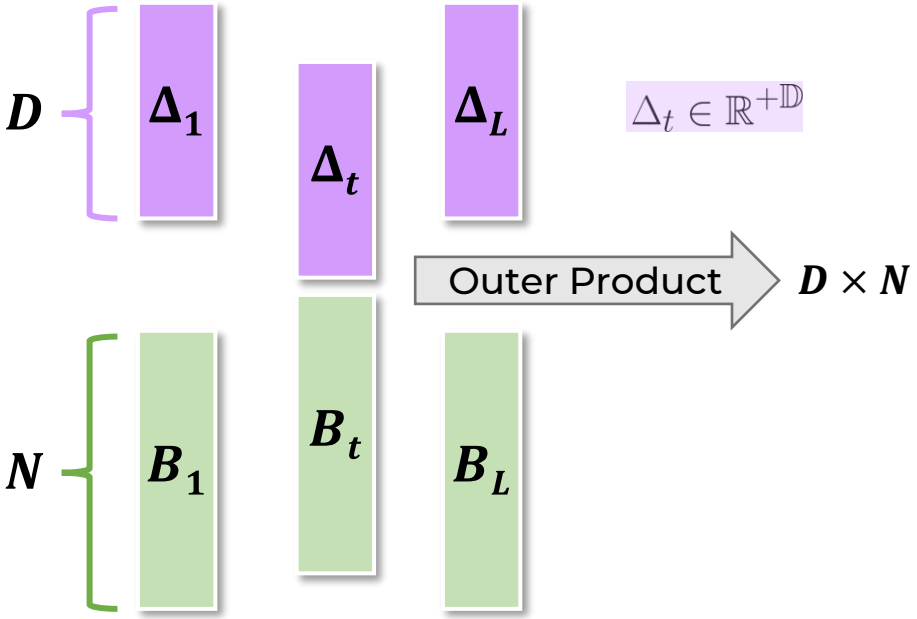
Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$

$\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_B(x)$
$\boldsymbol{C} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_C(x)$
$\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$ $\tau_\Delta = \text{softplus}$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
return y

Mamba

Welcome, Mamba!

- Improving SSMs with Selection



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

$\overline{B}$: 각 문장(Each B), 각 토큰의(Each L)
각 Input Scalar(Each D)에 대한 N projection

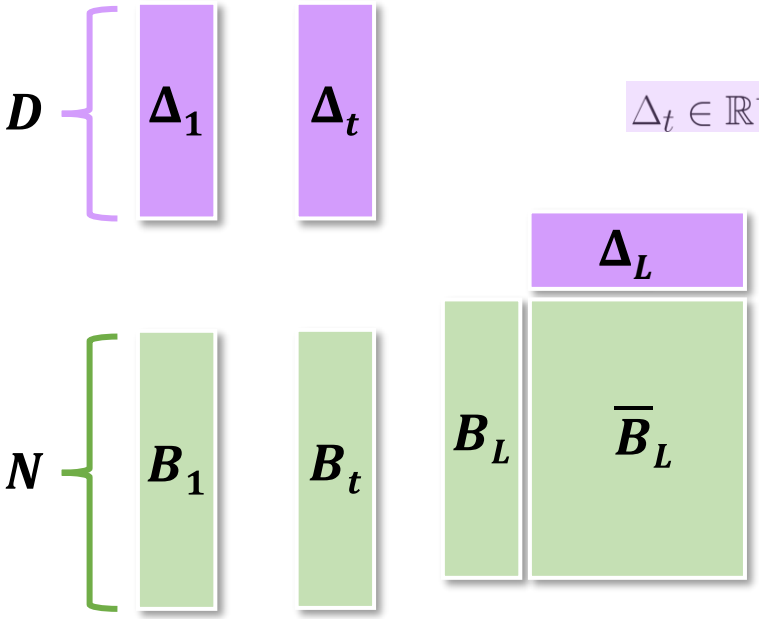
Algorithm 2 SSM + Selection (S6)

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

Mamba

Welcome, Mamba!

- Improving SSMs with Selection



$\Delta_t \in \mathbb{R}^{+\mathbb{D}}$

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

$\overline{B}$: 각 문장(Each B), 각 토큰의(Each L)
각 Input Scalar(Each D)에 대한 N projection

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: $\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$ $\tau_\Delta = \text{softplus}$
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

$\overline{B} = \exp(\Delta B)$
For Simplicity!

Mamba

Welcome, Mamba!

- Improving SSMs with Selection

ZOH Discretization

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

Δ → 0:
현재 Hidden State 유지
현재 Input 무시

Δ → ∞:
현재 Input에 대한
영향 증가

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: $\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$ $\tau_\Delta = \text{softplus}$
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

Mamba

Welcome, Mamba!

- Improving SSMs with Selection

Mamba내의 한 SSM Block의 디자인은 이러함

전체 Model Architecture는?

Algorithm 2 SSM + Selection (S6)

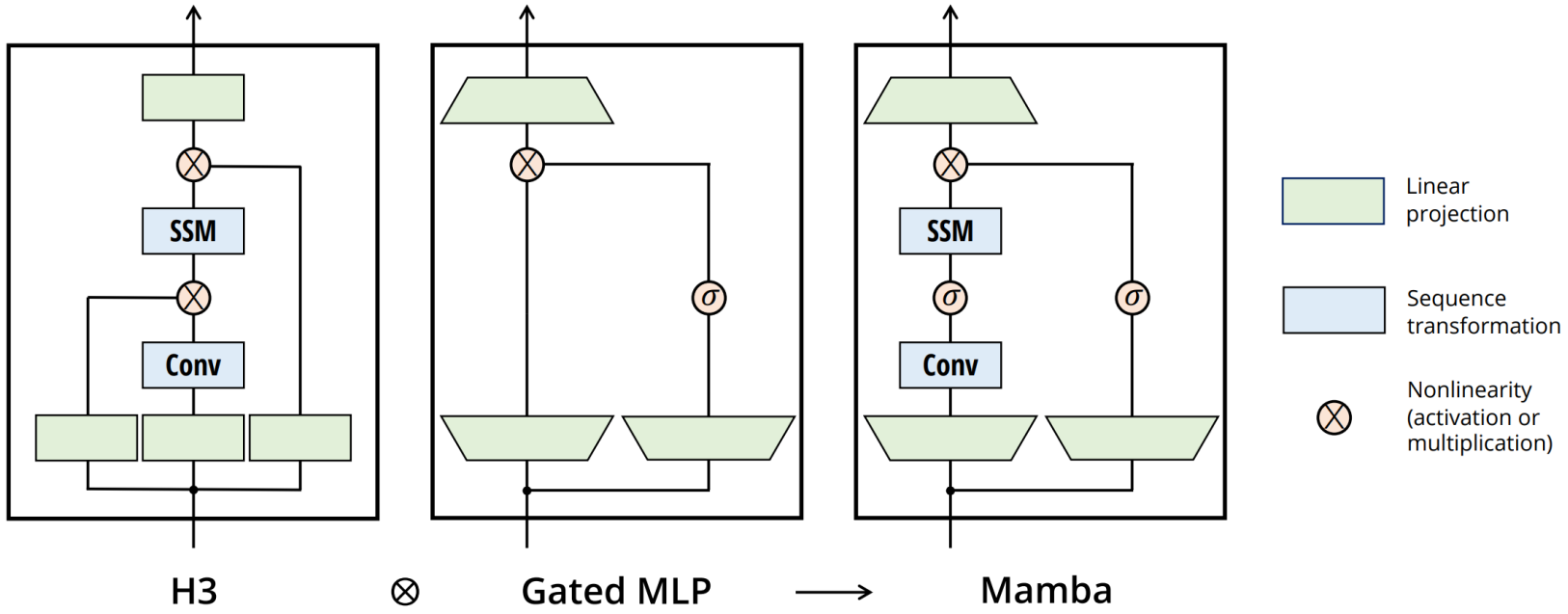
Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$

$\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_B(x)$
$\boldsymbol{C} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_C(x)$
$\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$ $\tau_\Delta = \text{softplus}$
$\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}} : (\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N}) \leftarrow \text{discretize}(\Delta, \boldsymbol{A}, \boldsymbol{B})$
$y \leftarrow \text{SSM}(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C})(x)$
▷ Time-varying: recurrence (*scan*) only
return y

Mamba

Welcome, Mamba!

- Overall Architecture

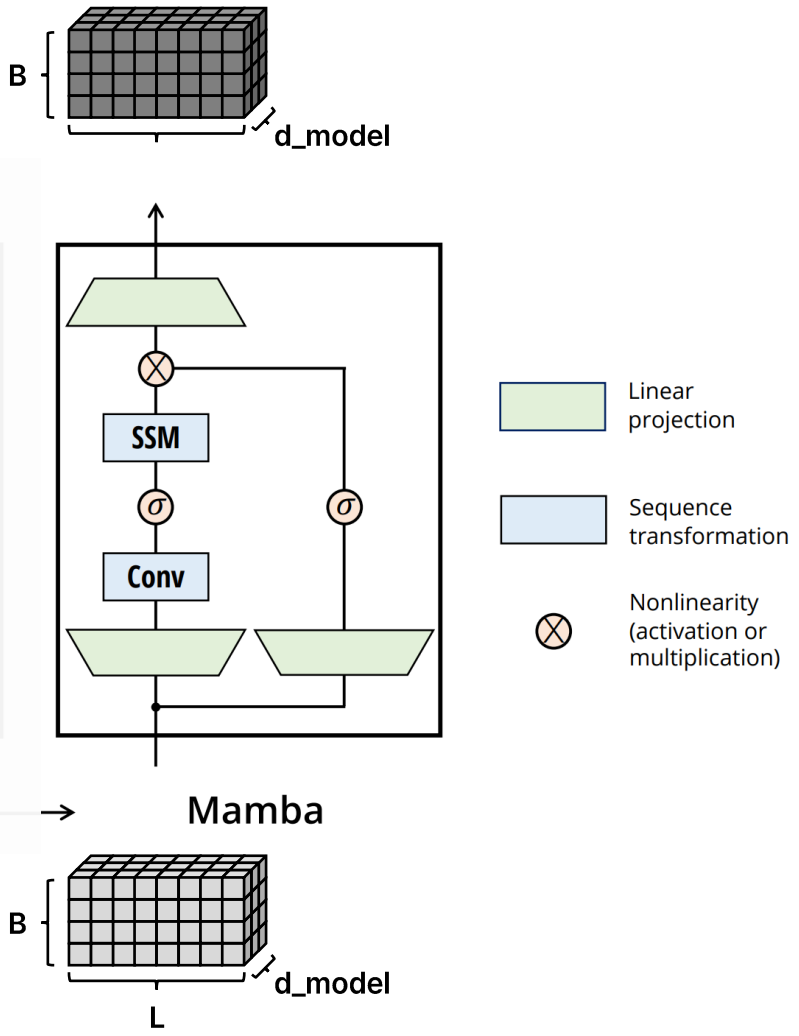


Mamba

Welcome, Mamba!

- Overall Architecture

B: Batch size
L: Sequence length
d_model: Block-to-block dimension

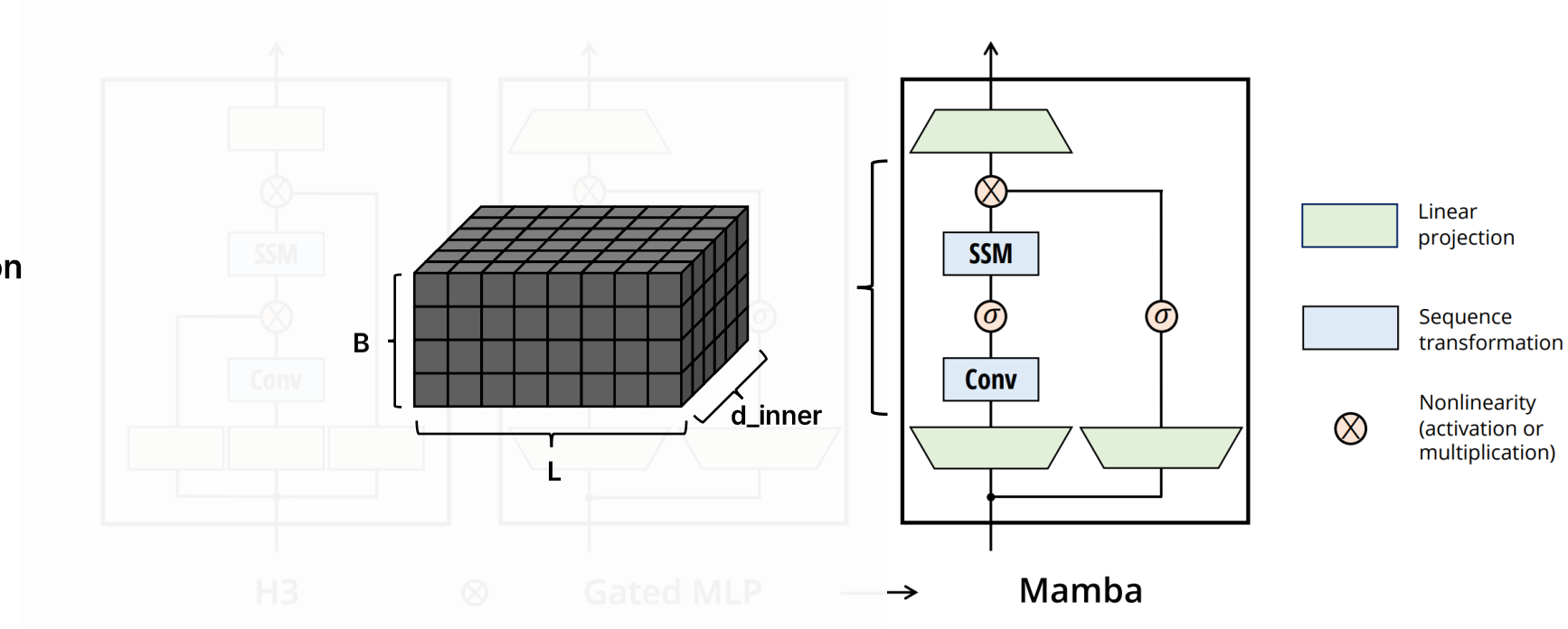


Mamba

Welcome, Mamba!

- Overall Architecture

B: Batch size
L: Sequence length
d_model: Block-to-block dimension
d_inner: 앞서 본 D와 동일



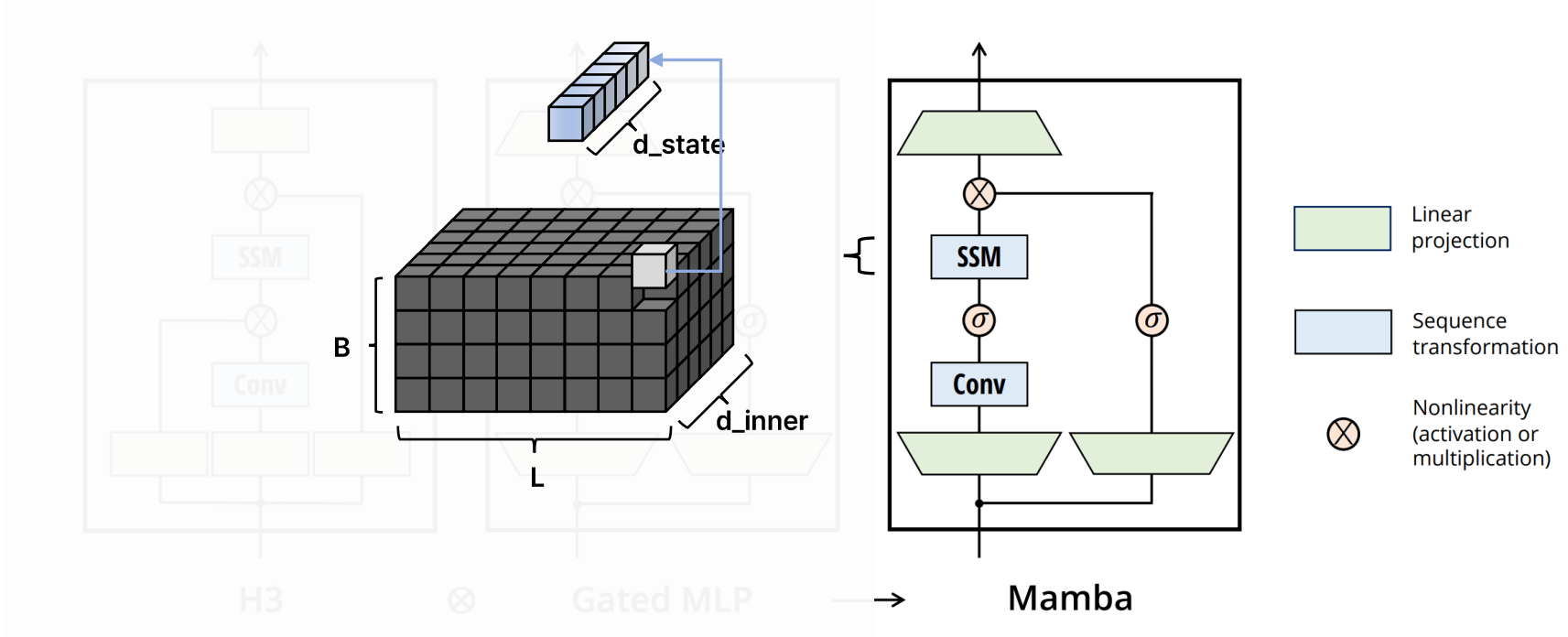
Mamba

Welcome, Mamba!

- Overall Architecture

B: Batch size
L: Sequence length
d_model: Block-to-block dimension
d_inner: 앞서 본 D와 동일
d_state: 앞서 본 N과 동일

d_inner개의 input variables에 대해
각각 d_state개의 hidden state가 존재

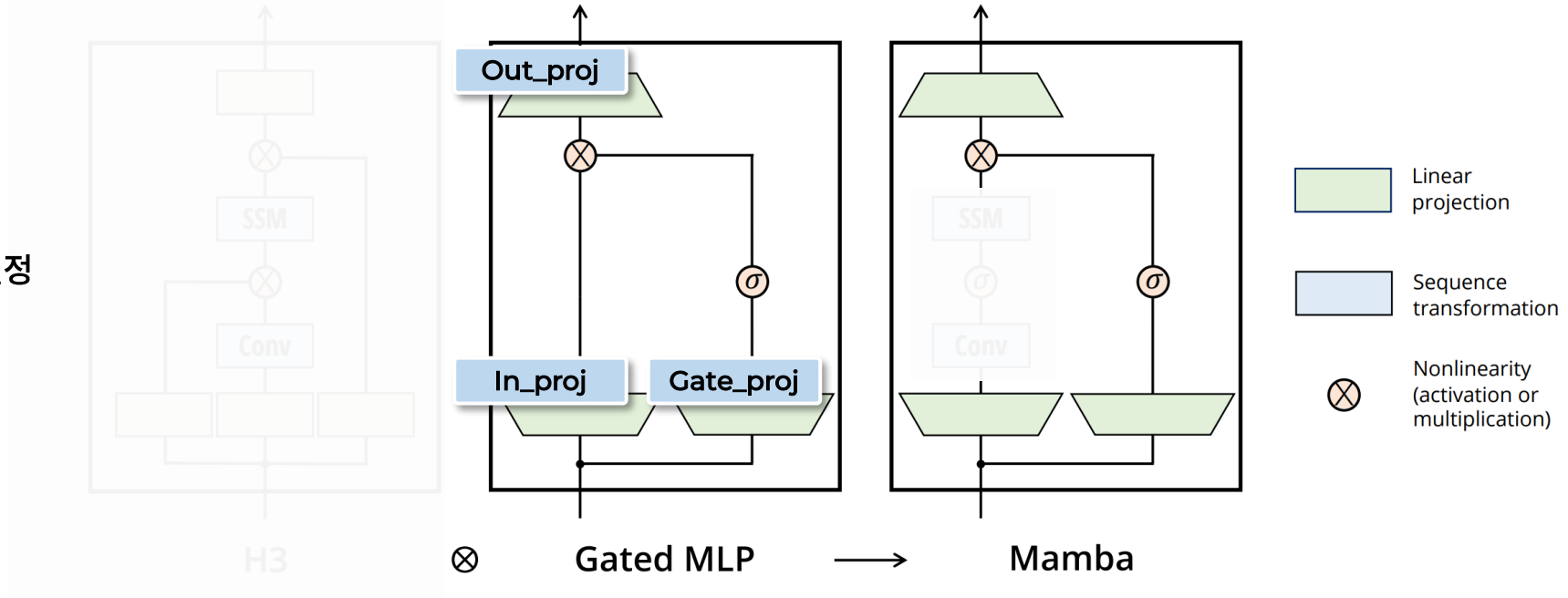


Mamba

Welcome, Mamba!

- Overall Architecture

Gate_proj: 어떤 Element를 걸러낼지 결정

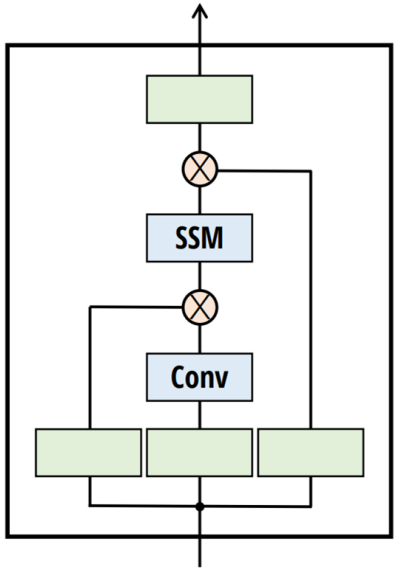


Mamba

Welcome, Mamba!

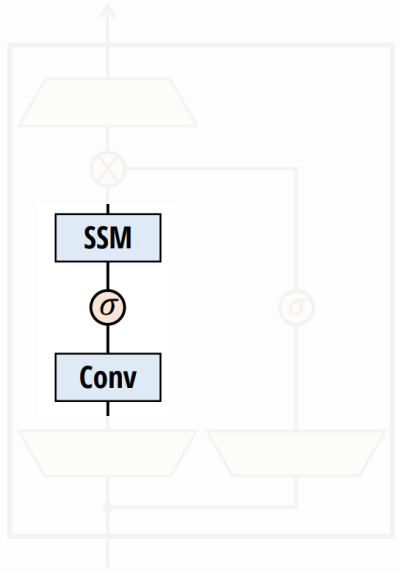
- Overall Architecture

Conv: Sequence 방향으로 Channel Mixing
d_inner dimension끼리는 Independent



H3

⊗ Gated MLP

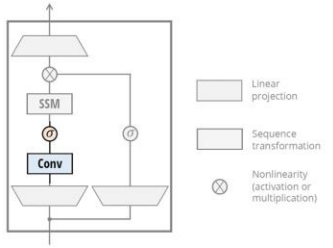


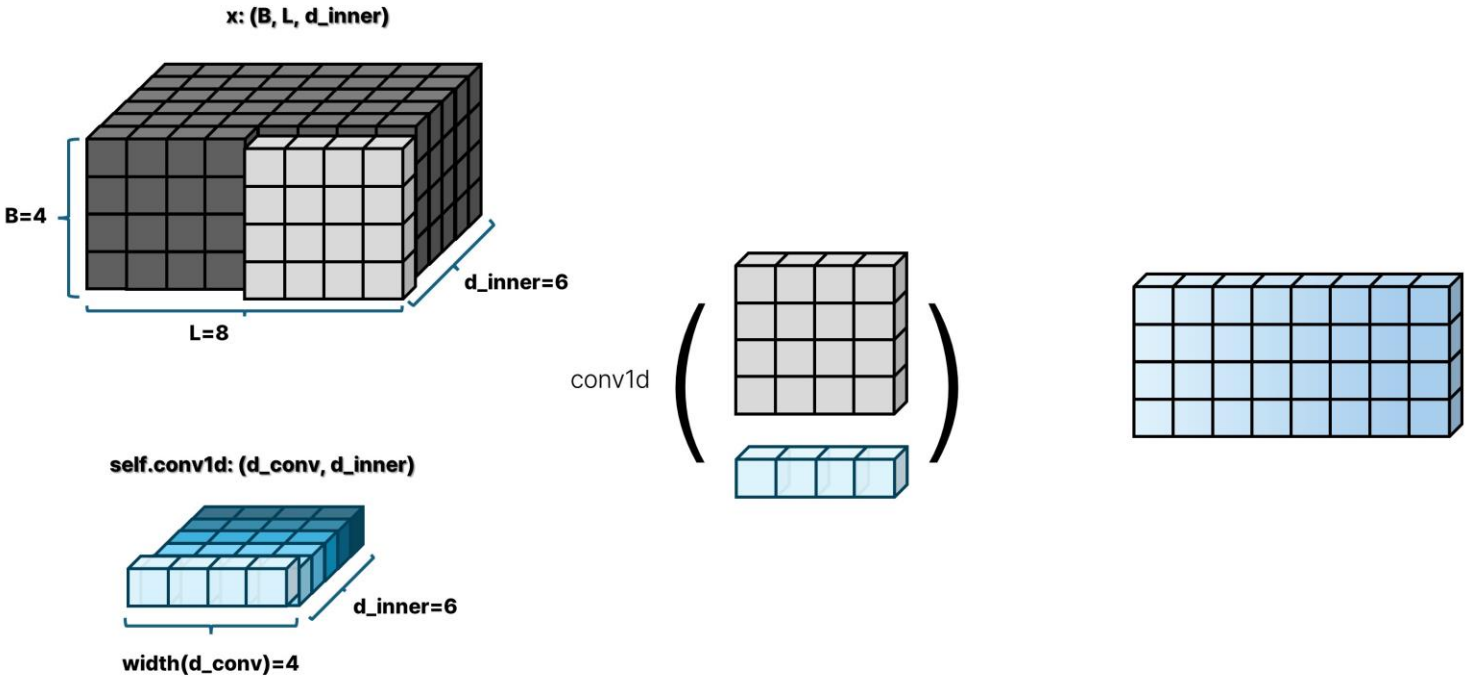
Mamba

Mamba

Welcome, Mamba!

- github.com/state-spaces/mamba/blob/main/mamba_ssm/modules/mamba_simple.py#L163~L177





Conv: S
d_inner

Linear projection
Sequence transformation
Nonlinearity (activation or multiplication)

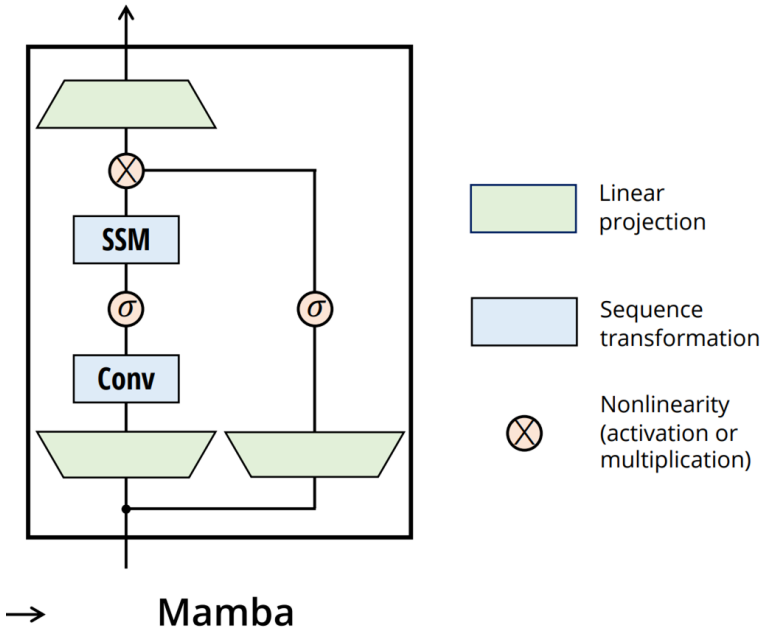
Mamba

Welcome, Mamba!

- Overcome the Computational Bottleneck

이제 좀 Mamba라는 걸 train시켜봐도 될까요...?

알아 두셔야 할 게 있습니다
좀 복잡하긴 한데요... 🤔

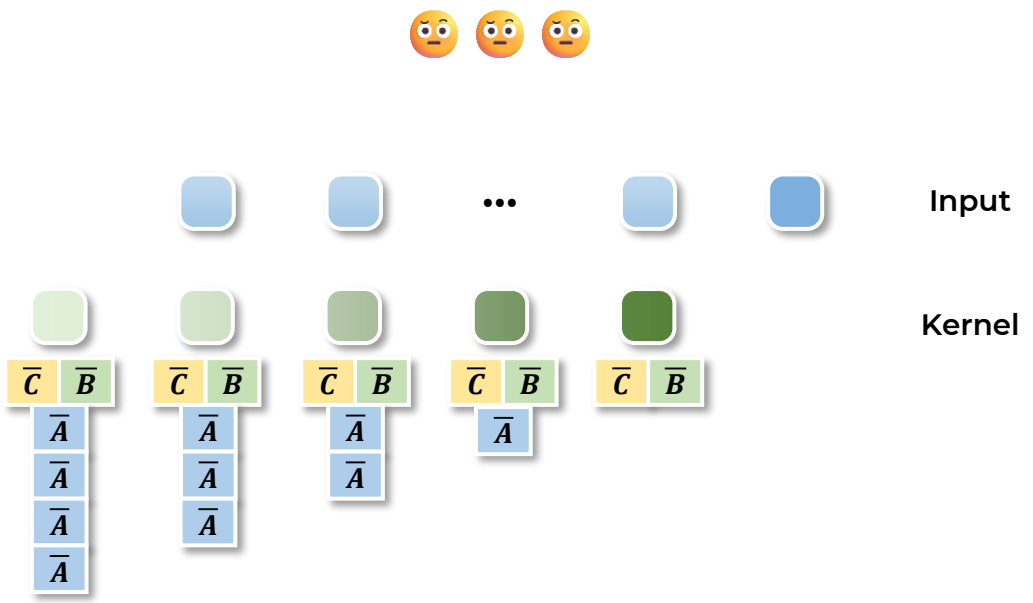


→ Mamba

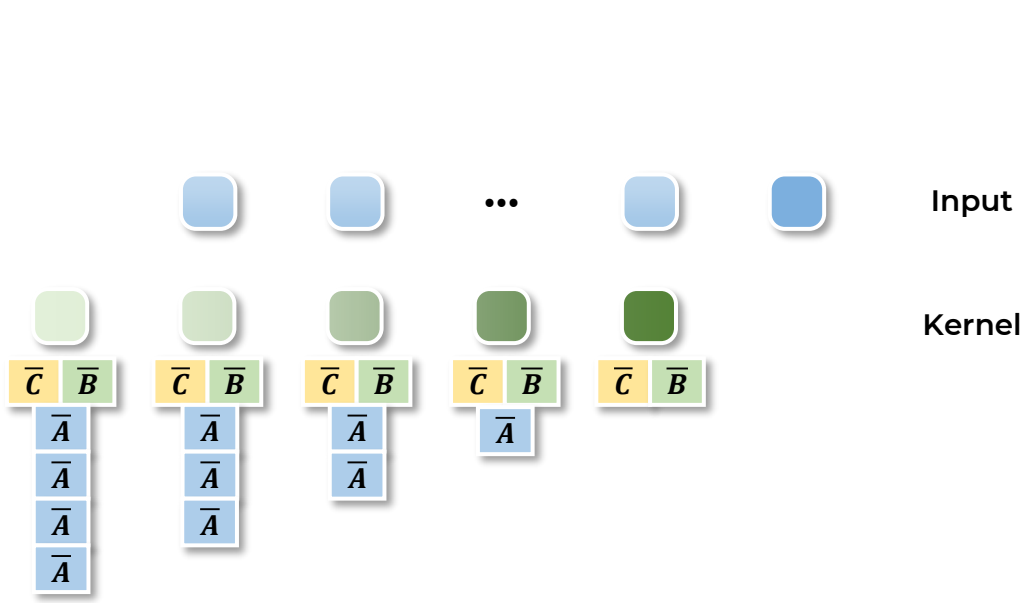
Welcome, Mamba!

- Overcome the Computational Bottleneck
  - ✓ Let's Recap SSM Convolution
    - 여전히 Training Parallelism이 적용되는가?

[Mamba Convolution]



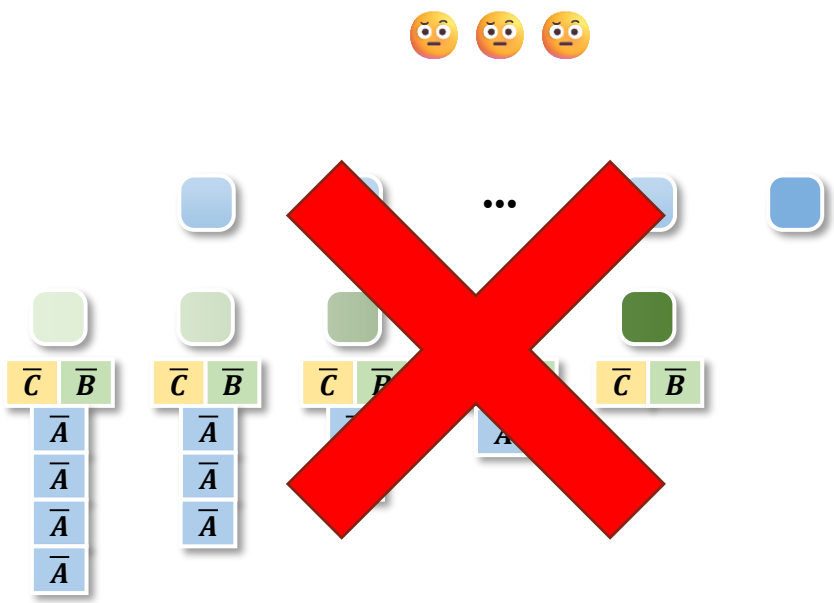
[SSM Convolution]



Welcome, Mamba!

- Overcome the Computational Bottleneck
  - ✓ Let's Recap SSM Convolution
    - 여전히 Training Parallelism이 적용되는가?

[Mamba Convolution]



System Matrices들이 더 이상 Input-Independent 하지 않음

미리 Global하게 적용될 수 있는 Kernel을 만드는 것이 불가

Convolution을 통한 Training Parallelism은 사용할 수 없음

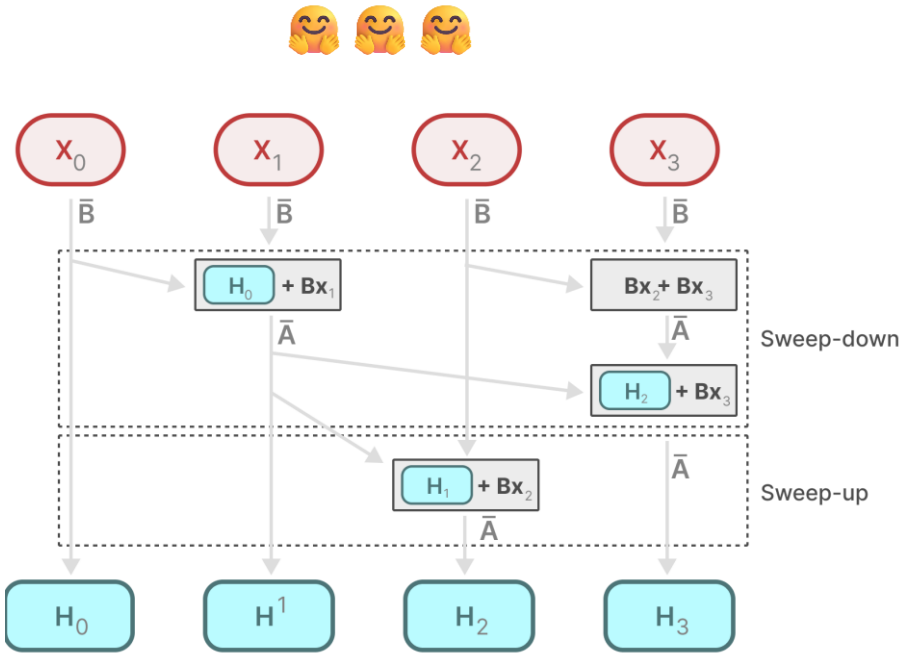
Any Ideas?

Mamba

Welcome, Mamba!

- Overcome the Computational Bottleneck
  - ✓ Use Parallel Scan, Instead!
    - 연산의 Associativity를 이용하자!

[Mamba Parallel Scan]



Parallel computation **O(n/t)**

Welcome, Mamba!

- Overcome the Computational Bottleneck
  - ✓ Use Parallel Scan, Instead!
    - 연산의 Associativity를 이용하자!

$$\mathbf{H}_0 = \overline{\mathbf{B}}_0\mathbf{x}_0$$

$$\mathbf{H}_1 = \overline{\mathbf{A}}_1(\overline{\mathbf{B}}_0\mathbf{x}_0) + \overline{\mathbf{B}}_1\mathbf{x}_1$$

$$= \overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{B}}_1\mathbf{x}_1$$

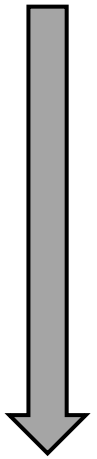
$$\mathbf{H}_2 = \overline{\mathbf{A}}_2(\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{B}}_1\mathbf{x}_1) + \overline{\mathbf{B}}_2\mathbf{x}_2$$

$$= \overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x}_1 + \overline{\mathbf{B}}_2\mathbf{x}_2$$

$$\mathbf{H}_3 = \overline{\mathbf{A}}_3(\overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x}_1 + \overline{\mathbf{B}}_2\mathbf{x}_2) + \overline{\mathbf{B}}_3\mathbf{x}_3$$

$$= \overline{\mathbf{A}}_3\overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{A}}_3\overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x}_1 + \overline{\mathbf{A}}_3\overline{\mathbf{B}}_2\mathbf{x}_2 + \overline{\mathbf{B}}_3\mathbf{x}_3$$

Looks Like $O(n)$...
Cannot Compute H_t Before H_{t-1}



Welcome, Mamba!

- Overcome the Computational Bottleneck
  - ✓ Use Parallel Scan, Instead!
    - 연산의 Associativity를 이용하자!

Looks Like $O(n)$...
Cannot Compute H_t Before H_{t-1}

$$\mathbf{H}_0 = \overline{\mathbf{B}}_0\mathbf{x}_0$$

$$\mathbf{H}_1 = \overline{\mathbf{A}}_1(\overline{\mathbf{B}}_0\mathbf{x}_0) + \overline{\mathbf{B}}_1\mathbf{x}_1$$

$$= \overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{B}}_1\mathbf{x}_1$$

$$\mathbf{H}_2 = \overline{\mathbf{A}}_2(\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{B}}_1\mathbf{x}_1) + \overline{\mathbf{B}}_2\mathbf{x}_2$$

$$= \overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x}_1 + \overline{\mathbf{B}}_2\mathbf{x}_2$$

$$\mathbf{H}_3 = \overline{\mathbf{A}}_3(\overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x}_1 + \overline{\mathbf{B}}_2\mathbf{x}_2) + \overline{\mathbf{B}}_3\mathbf{x}_3$$

$$= \overline{\mathbf{A}}_3\overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{A}}_3\overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x}_1 + \overline{\mathbf{A}}_3\overline{\mathbf{B}}_2\mathbf{x}_2 + \overline{\mathbf{B}}_3\mathbf{x}_3$$

Welcome, Mamba!

- Overcome the Computational Bottleneck
  - ✓ Use Parallel Scan, Instead!
    - 연산의 Associativity를 이용하자!

Looks Like $O(n)$...
Cannot Compute H_t Before H_{t-1}

$$\mathbf{H_0} = \overline{\mathbf{B}}_0\mathbf{x_0}$$

$$\mathbf{H_1} = \overline{\mathbf{A}}_1(\overline{\mathbf{B}}_0\mathbf{x_0}) + \overline{\mathbf{B}}_1\mathbf{x_1}$$

$$= \overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x_0} + \overline{\mathbf{B}}_1\mathbf{x_1}$$

$$\mathbf{H_2} = \overline{\mathbf{A}}_2(\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x_0} + \overline{\mathbf{B}}_1\mathbf{x_1}) + \overline{\mathbf{B}}_2\mathbf{x_2}$$

$$= \overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x_0} + \overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x_1} + \overline{\mathbf{B}}_2\mathbf{x_2}$$

$$\mathbf{H_3} = \overline{\mathbf{A}}_3(\overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x_0} + \overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x_1} + \overline{\mathbf{B}}_2\mathbf{x_2}) + \overline{\mathbf{B}}_3\mathbf{x_3}$$

$$= \overline{\mathbf{A}}_3\overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x_0} + \overline{\mathbf{A}}_3\overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x_1} + \overline{\mathbf{A}}_3\overline{\mathbf{B}}_2\mathbf{x_2} + \overline{\mathbf{B}}_3\mathbf{x_3}$$

Welcome, Mamba!

- Overcome the Computational Bottleneck
  - ✓ Use Parallel Scan, Instead!
    - 연산의 Associativity를 이용하자!

$$\mathbf{H_0} = \overline{\mathbf{B}}_0\mathbf{x_0}$$

$$\mathbf{H_1} = \overline{\mathbf{A}}_1(\overline{\mathbf{B}}_0\mathbf{x_0}) + \overline{\mathbf{B}}_1\mathbf{x_1}$$

$$= \overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x_0} + \overline{\mathbf{B}}_1\mathbf{x_1}$$

$$\mathbf{H_2} = \overline{\mathbf{A}}_2(\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x_0} + \overline{\mathbf{B}}_1\mathbf{x_1}) + \overline{\mathbf{B}}_2\mathbf{x_2}$$

$$= \overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x_0} + \overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x_1} + \overline{\mathbf{B}}_2\mathbf{x_2}$$

$$\mathbf{H_3} = \overline{\mathbf{A}}_3(\overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x_0} + \overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x_1} + \overline{\mathbf{B}}_2\mathbf{x_2}) + \overline{\mathbf{B}}_3\mathbf{x_3}$$

$$= \overline{\mathbf{A}}_3\overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x_0} + \overline{\mathbf{A}}_3\overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x_1} + \overline{\mathbf{A}}_3\overline{\mathbf{B}}_2\mathbf{x_2} + \overline{\mathbf{B}}_3\mathbf{x_3}$$

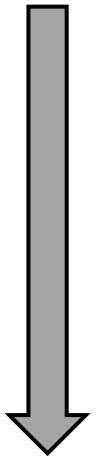
Looks Like $O(n)$...
Cannot Compute H_t Before H_{t-1}

Welcome, Mamba!

- Overcome the Computational Bottleneck
  - ✓ Use Parallel Scan, Instead!
    - 연산의 Associativity를 이용하자!

Looks Like $O(n)$...
Cannot Compute H_t Before H_{t-1}

$$
\begin{aligned}
\mathbf{H_0} &= \overline{\mathbf{B}}_0\mathbf{x}_0 \\
\mathbf{H_1} &= \overline{\mathbf{A}}_1(\overline{\mathbf{B}}_0\mathbf{x}_0) + \overline{\mathbf{B}}_1\mathbf{x}_1 \\
&= \overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{B}}_1\mathbf{x}_1 \\
\mathbf{H_2} &= \overline{\mathbf{A}}_2(\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{B}}_1\mathbf{x}_1) + \overline{\mathbf{B}}_2\mathbf{x}_2 \\
&= \overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x}_1 + \overline{\mathbf{B}}_2\mathbf{x}_2 \\
\mathbf{H_3} &= \overline{\mathbf{A}}_3(\overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x}_1 + \overline{\mathbf{B}}_2\mathbf{x}_2) + \overline{\mathbf{B}}_3\mathbf{x}_3 \\
&= \overline{\mathbf{A}}_3\overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{A}}_3\overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x}_1 + \overline{\mathbf{A}}_3\overline{\mathbf{B}}_2\mathbf{x}_2 + \overline{\mathbf{B}}_3\mathbf{x}_3 \\
&\dots
\end{aligned}
$$



Welcome, Mamba!

- Overcome the Computational Bottleneck
  - ✓ Use Parallel Scan, Instead!
    - 연산의 Associativity를 이용하자!

But, Maybe We can Compute Something Else...

$$
\begin{aligned}
\mathbf{H}_0 &= \overline{\mathbf{B}}_0\mathbf{x}_0 \\
\mathbf{H}_1 &= \overline{\mathbf{A}}_1(\overline{\mathbf{B}}_0\mathbf{x}_0) + \overline{\mathbf{B}}_1\mathbf{x}_1 \\
&= \overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{B}}_1\mathbf{x}_1 \\
\mathbf{H}_2 &= \overline{\mathbf{A}}_2(\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{B}}_1\mathbf{x}_1) + \overline{\mathbf{B}}_2\mathbf{x}_2 \\
&= \overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x}_1 + \overline{\mathbf{B}}_2\mathbf{x}_2 \\
\mathbf{H}_3 &= \overline{\mathbf{A}}_3(\overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x}_1 + \overline{\mathbf{B}}_2\mathbf{x}_2) + \overline{\mathbf{B}}_3\mathbf{x}_3 \\
&= \overline{\mathbf{A}}_3\overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{A}}_3\overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x}_1 + \overline{\mathbf{A}}_3\overline{\mathbf{B}}_2\mathbf{x}_2 + \overline{\mathbf{B}}_3\mathbf{x}_3
\end{aligned}
$$

Welcome, Mamba!

- Overcome the Computational Bottleneck
  - ✓ Use Parallel Scan, Instead!
    - 연산의 Associativity를 이용하자!

$$\mathbf{H}_0 = \overline{\mathbf{B}}_0\mathbf{x}_0$$

$$\begin{aligned}\mathbf{H}_1 &= \overline{\mathbf{A}}_1(\overline{\mathbf{B}}_0\mathbf{x}_0) + \overline{\mathbf{B}}_1\mathbf{x}_1 \\ &= \overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{B}}_1\mathbf{x}_1\end{aligned}$$

$$\begin{aligned}\mathbf{H}_2 &= \overline{\mathbf{A}}_2(\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{B}}_1\mathbf{x}_1) + \overline{\mathbf{B}}_2\mathbf{x}_2 \\ &= \overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x}_1 + \overline{\mathbf{B}}_2\mathbf{x}_2\end{aligned}$$

$$\begin{aligned}\mathbf{H}_3 &= \overline{\mathbf{A}}_3(\overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x}_1 + \overline{\mathbf{B}}_2\mathbf{x}_2) + \overline{\mathbf{B}}_3\mathbf{x}_3 \\ &= \overline{\mathbf{A}}_3\overline{\mathbf{A}}_2\overline{\mathbf{A}}_1\overline{\mathbf{B}}_0\mathbf{x}_0 + \overline{\mathbf{A}}_3\overline{\mathbf{A}}_2\overline{\mathbf{B}}_1\mathbf{x}_1 + \overline{\mathbf{A}}_3\overline{\mathbf{B}}_2\mathbf{x}_2 + \overline{\mathbf{B}}_3\mathbf{x}_3\end{aligned}$$

Compute Matrices Multiplication with Multiple Threads

Complexity Drops to $O(n/T)$

Welcome, Mamba!

- Overcome the Computational Bottleneck
  - ✓ Still, Lots of Tensor Operations Needed
    - Those Dimensionalities!

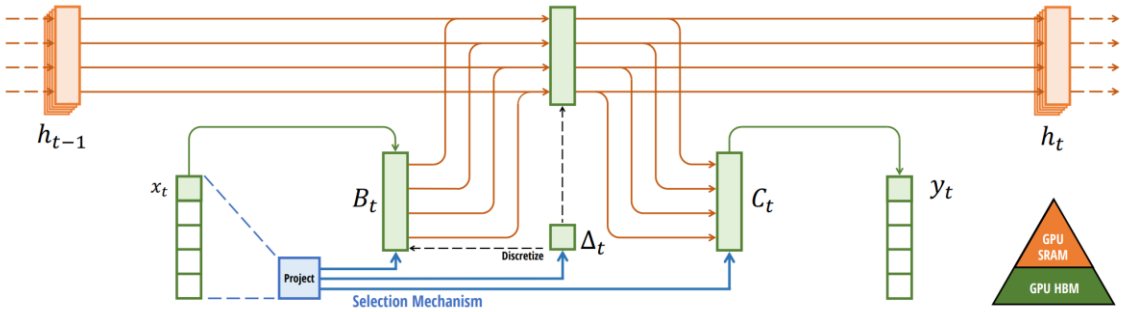
B개의 Sequences 속의
L개의 tokens들 속의
D개의 Input variables들 속의
N차원의 Latent space를 모델링

B x L x D의 세 차원으로 운용되는 Transformer에 비해
GPU적인 부담이 큼

Mamba

Welcome, Mamba!

- Overcome the Computational Bottleneck
  - ✓ Kernel Fusion
    - The Bottleneck is all about Copy & Paste!
    - Matmul을 제외한 모든 연산은 사실상
      실제 SRAM에서의 연산보다
      HBM에서 SRAM으로 복사하는 것이 오래 걸림



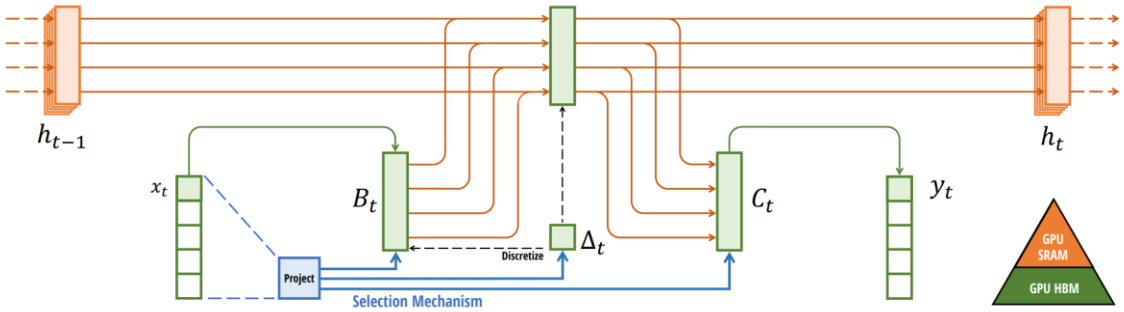
The main idea is to leverage properties of modern accelerators (GPUs) to materialize the state h only in more efficient levels of the memory hierarchy. In particular, most operations (except matrix multiplication) are bounded by memory bandwidth (Dao, Fu, Ermon, et al. 2022; Ivanov et al. 2021; Williams, Waterman, and Patterson 2009). This includes our scan operation, and we use kernel fusion to reduce the amount of memory IOs, leading to a significant speedup compared to a standard implementation.

Mamba

Welcome, Mamba!

- Overcome the Computational Bottleneck
  - ✓ Kernel Fusion

3개의 dimension을 가진 parameters들을
From HBM To SRAM



Algorithm 2 SSM + Selection (S6)

Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$

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

Mamba

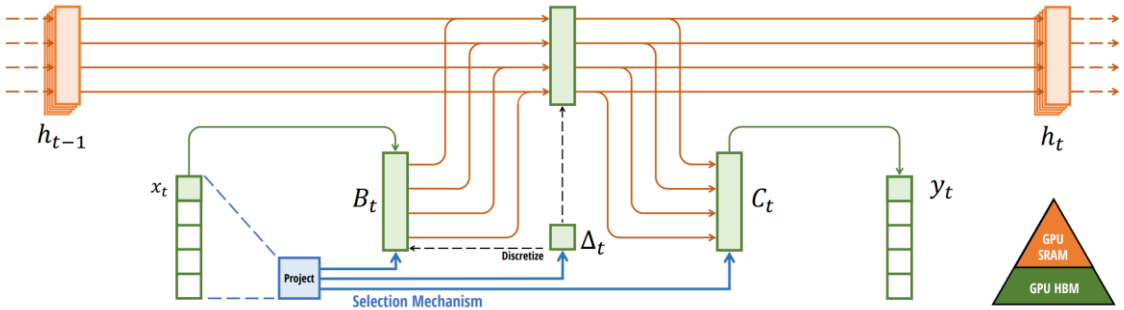
Welcome, Mamba!

- Overcome the Computational Bottleneck

  ✓ Kernel Fusion

3개의 dimension을 가진 parameters들을
From HBM To SRAM

차원이 늘어나기 시작하는 Discretization을
SRAM에서 바로 진행



Algorithm 2 SSM + Selection (S6)

Input: x : (B, L, D)
Output: y : (B, L, D)

1: $\boldsymbol{A}$: (D, N) ← Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B}$: (B, L, N) ← $s_B(x)$
3: $\boldsymbol{C}$: (B, L, N) ← $s_C(x)$
4: Δ : (B, L, D) ← τ_Δ(Parameter+$s_\Delta(x)$)
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}$: (B, L, D, N) ← 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

Mamba

Welcome, Mamba!

- Overcome the Computational Bottleneck

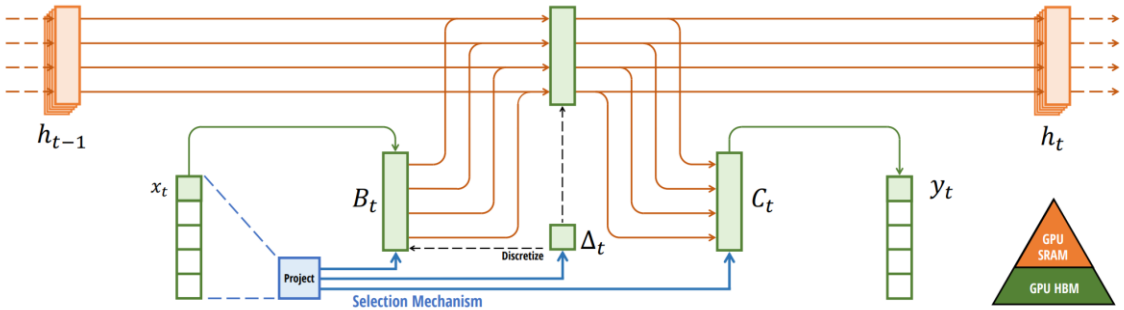
  ✓ Kernel Fusion

3개의 dimension을 가진 parameters들을
From HBM To SRAM

차원이 늘어나기 시작하는 Discretization을
SRAM에서 바로 진행

SSM 후 차원이 줄어든 결과물 y를
From SRAM to HBM

O(N)만큼의 IOs를 줄이게 됨



Algorithm 2 SSM + Selection (S6)

Input: x : (B, L, D)
Output: y : (B, L, D)
1: $\boldsymbol{A}$: (D, N) ← Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B}$: (B, L, N) ← $s_B(x)$
3: $\boldsymbol{C}$: (B, L, N) ← $s_C(x)$
4: Δ : (B, L, D) ← τ_Δ(Parameter+$s_\Delta(x)$)
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}$: (B, L, D, N) ← 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

Delve into Mamba!

Delve into Mamba!

Mamba

Delve into Mamba!

- What does “Selective” mean?
  - ✓ 단순히 Input을 처리하는 것이 아닌, Input을 처리하는 Transition 자체가 Input에 따라(along the sequence length) 변해야 함을 의미
  - ✓ Input에 따라 Transition을 만들어낼 수 있는 System을 학습시켜야 함

A Discussion: Selection Mechanism

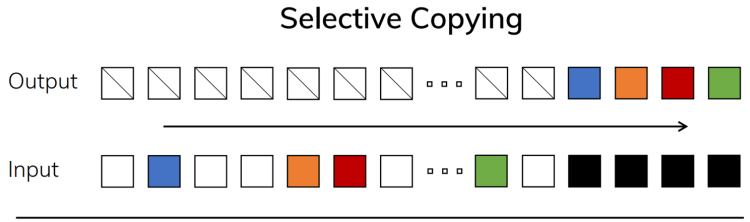
⋮

“gating” in favor of selection to clarify the overloaded use of former. More narrowly, we use selection to refer to the mechanistic action of a model to select or ignore inputs and facilitate data interaction along the sequence length (Section 3.1). Beyond selective SSMs and gated RNNs, other examples may include input-dependent convolutions (Kosma, Nikolentzos, and Vazirgiannis 2023; Lioutas and Guo 2020; Lutati, Zimerman, and Wolf 2023; Yang et al. 2019) and even attention.

Mamba

Delve into Mamba!

- What does “Selective” mean?
  - ✓ 실험 결과, Mamba의 Model이나 Architecture보다도, Layer가 Selective Copying task에서 중요했음
    - (c.f. H3)
  - ✓ 물론, Architecture적으로도 Mamba가 더 좋음
  - ✓ Selective Copying task 역시, synthetic하게 만들어낸 실험 환경



Selective Copying. Our setting is on sequences of length 4096, with a vocab size of 16 possible tokens (including the white “noise” token from Figure 2) and requiring models to memorize 16 “data” tokens. We use 2 layer models with a model dimension of $D = 64$.

Models are trained for 400K steps at a constant learning rate of 0.0001 with a batch size of 64.

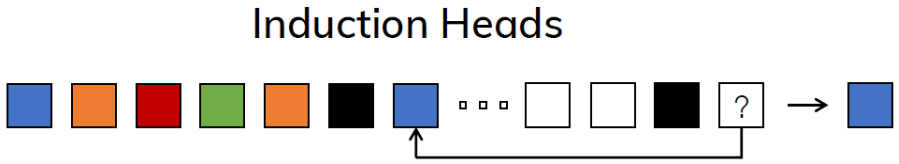
Model	Arch.	Layer	Acc.
S4	No gate	S4	18.3
-	No gate	S6	**97.0**
H3	H3	S4	57.0
Hyena	H3	Hyena	30.1
-	H3	S6	**99.7**
-	Mamba	S4	56.4
-	Mamba	Hyena	28.4
Mamba	Mamba	S6	**99.8**

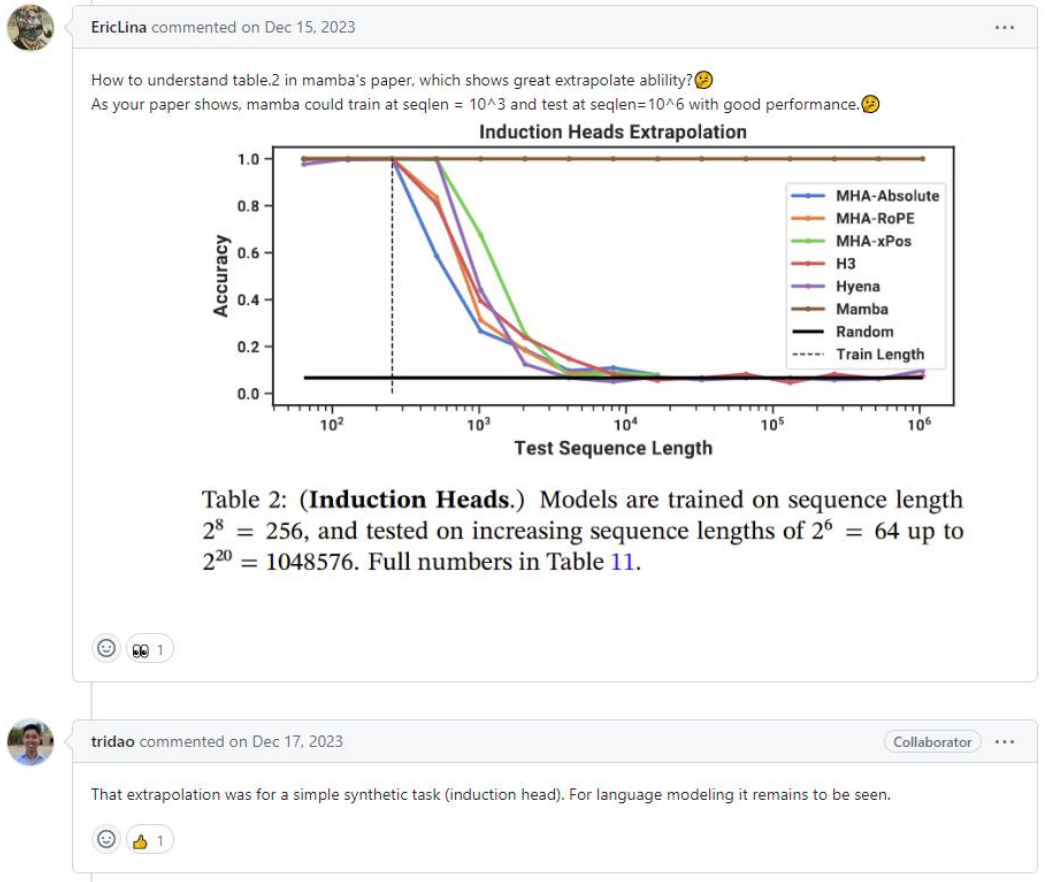
Table 1: (**Selective Copying.**) Accuracy for combinations of architectures and inner sequence layers.

Mamba

Delve into Mamba!

- What does “Selective” mean?
  - ✓ Induction Head task 또한, Selectiveness가 중요할 것이라고 주장했었음
  - ✓ 특히나, Extrapolation에서 월등한 성능을 보임
  - ✓ Language Modelling에서의 Extrapolation의 성능이 아님을 유의

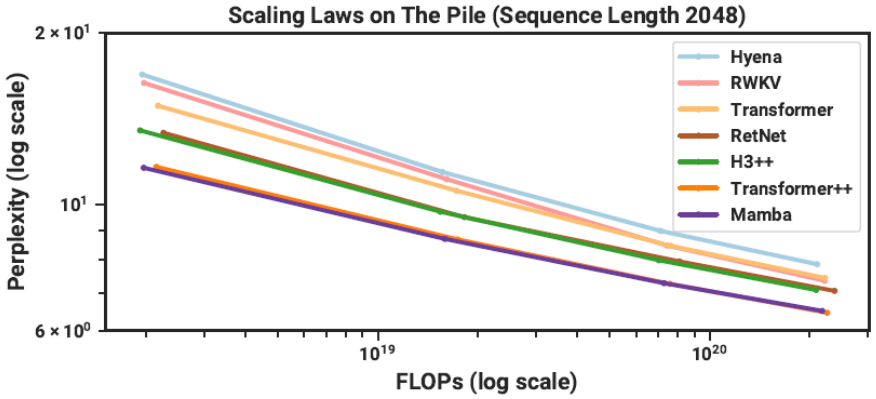


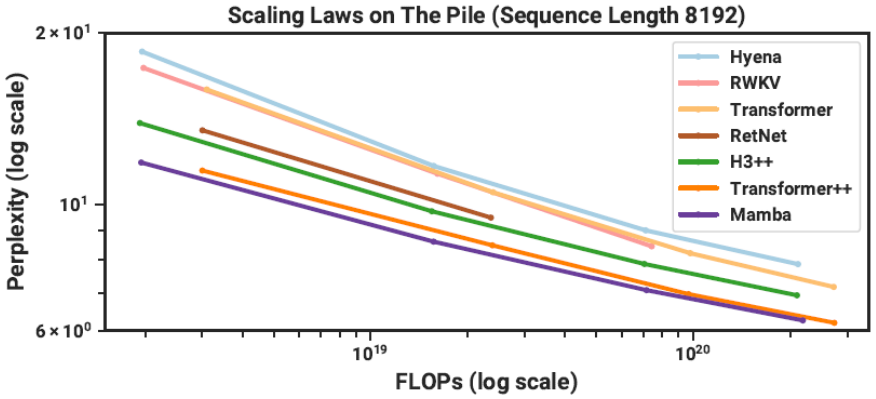


Mamba

Delve into Mamba!

- Finally, Knocking on a LM's door with SSM
  - ✓ High-end recipe로 만든 Transformer++은, 역시 Scaling Law를 잘 따르고 있음
  - ✓ Mamba 역시, 다른 attention-free 모델들에 비해 Transformer++정도의 Scaling Law를 보여줌





Mamba

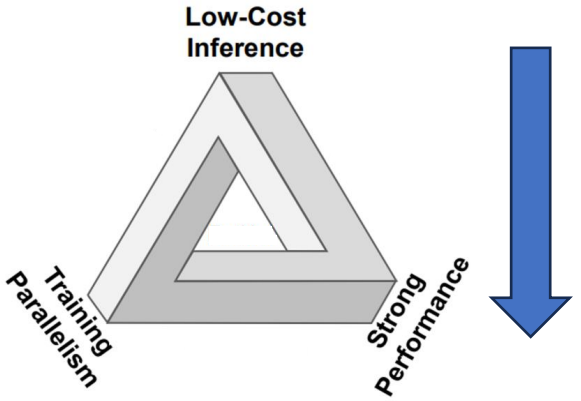
Delve into Mamba!

- Finally, Knocking on a LM's door with SSM
  - ✓ 비슷한 크기의 모델들과 Downstream task 성능 비교
  - ✓ Llama-2나 Mistral은 #params 차이로 제외
  - ✓ Downstream task도 #params 대비 나쁘지 않게 할 수 있다

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

Delve into Mamba!

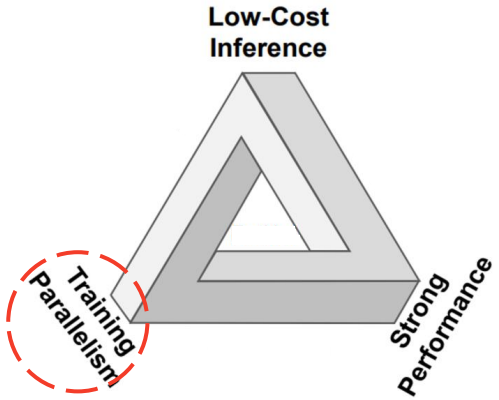
- What about the Other Sides of the Triangle?
  - ✓ Attention-Free의 자랑, Inference Time
  - ✓ Attention-Free의 숙제, Training Parallelism

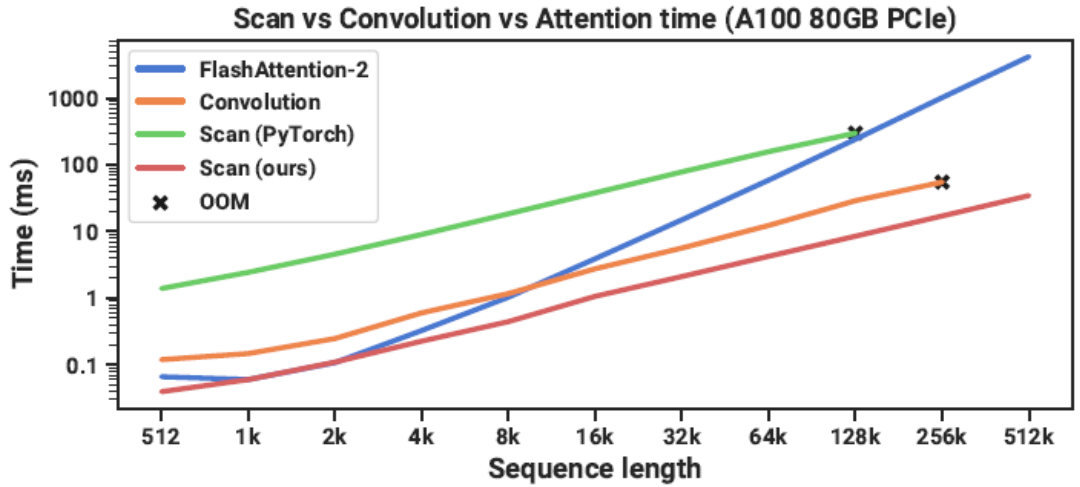


Mamba

Delve into Mamba!

- What about the Other Sides of the Triangle?
  - ✓ Attention-Free의 숙제, Training Parallelism
    - Even Faster than FlashAttention-2 over 2k!!!
    - Hardware-aware scan이 기본 scan에 비해서도 20-40×배 빠름

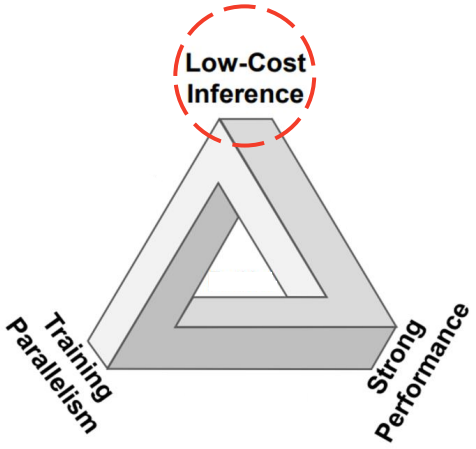


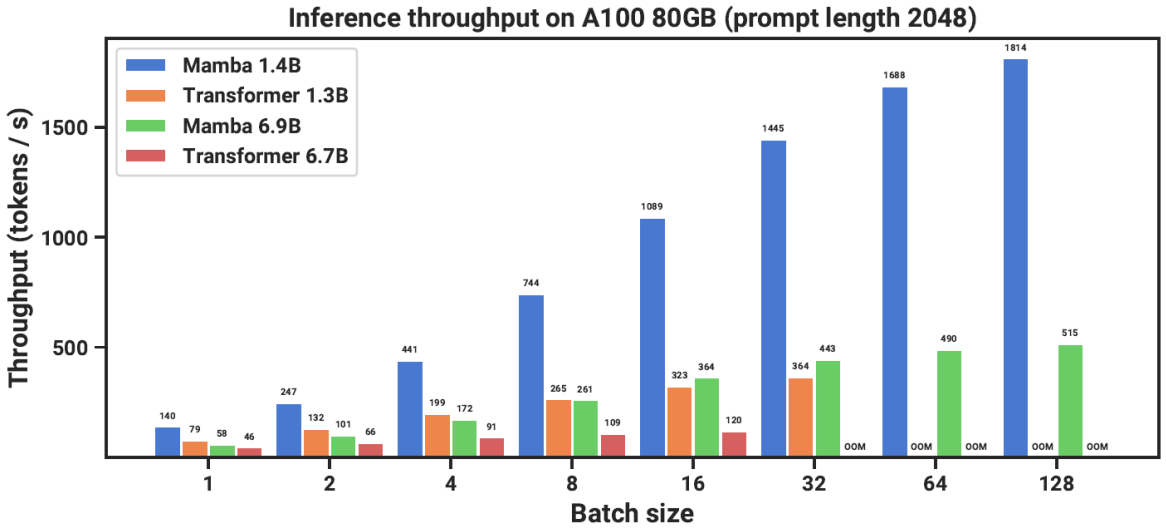


Mamba

Delve into Mamba!

- What about the Other Sides of the Triangle?
  - ✓ Attention-Free의 자랑, Inference Time
    - Mamba 6.9B가 TF 1.3B와 비슷할 정도





Reflect on Mamba!

Reflect on Mamba!

Preliminary

- Personal thought on Mamba

So, why do we have to use SSM?

Can be seen as over-engineering

Still, isn’t it patch-work?

Preliminary

- Personal thought on Mamba

Preliminary

- Personal thought on Mamba

Preliminary

- To wrap it up…

✓ Mamba는 Transformer보다 낮은, Sub-quadratic한 Time-complexity를 가지는 Sequential Model
  - structured SSM과 철학을 공유하며, 더 발전시킨 model

✓ 기존의 structured SSM이 본질적으로 'selectivity'라는 능력을 가질 수 없다는 것을 증명
  - 이를 보완해줌으로서 structured SSM의 성능을 높일 수 있음을 주장하고 증명함
  - Language에도 적용될 수 있을 정도로 Transformer급의 Performance를 보임

✓ 이 보완점을 위해 추가한 연산에 의해 SSM의 training parallelism의 핵심이었던 convolution 연산이 불가능해짐
  - Parallel associative scan을 통해 해결, 결론적으로 삼각형의 세 변 모두를 잡음

0. Sources I've Referenced

0.1 Papers

- PARALLELIZING LINEAR RECURRENT NEURAL NETS OVER SEQUENCE LENGTH
- Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks
- HiPPO: Recurrent Memory with Optimal Polynomial Projections
- Combining Recurrent, Convolutional, and Continuous-time Models with Linear State-Space Layers
- Hyena Hierarchy: Towards Larger Convolutional Language Models
- Retentive Network: A Successor to Transformer for Large Language Models
- Efficiently Modeling Long Sequences with Structured State Spaces
- On the Parameterization and Initialization of Diagonal State Space Models
- SIMPLIFIED STATE SPACE LAYERS FOR SEQUENCE MODELING
- Hungry Hungry Hippos: Towards Language Modeling with State Space Models
- FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

0.2 Codes & Visuals & Blogs

- https://github.com/state-spaces/s4
- https://github.com/state-spaces/mamba
- https://srush.github.io/annotated-s4/
- https://www.youtube.com/watch?v=8Q_tqwpTpVU
- https://www.youtube.com/watch?v=h_VpKMFrxN0
- https://velog.io/@stapers/Linear-Transformer

Thank you

감사합니다