Mamba: Linear-Time Sequence Modeling with Selective State Spaces

Albert Gu[*1] and Tri Dao[*2]

[1]Machine Learning Department, Carnegie Mellon University

[2]Department of Computer Science, Princeton University
agu@cs.cmu.edu, tri@tridao.me

[2025.09.25. Journal Club]

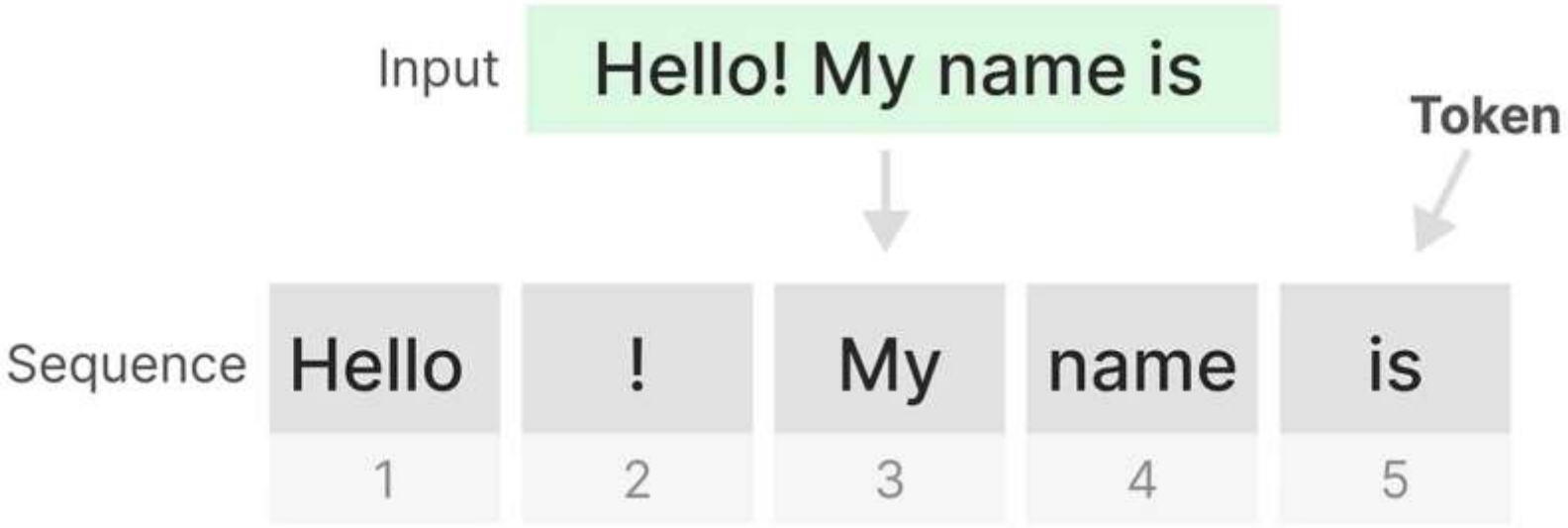
Computer Science and Engineering
Incheon National University
Visual Information Perception Laboratory MINJE JEON

MAMBA VS TRANSFORMER

Introduction

- Transformer는 context window 밖을 다루지 못하고 window 길이에 따라 계산이 이차적으로 늘어나는 한계가 있음.
- 기존 SSM들은 Audio and VISION 등 연속 Signal에는 강하지만 text 같은 이산.정보 밀집 data에는 약했다.
- 우리는 선택 메커니즘을 바탕으로 Attention, 별도의 MLP없이 반복되는 단일 Blocks으로 단순화한 Mamba를 제시했고, context 를 길게 가져갈수록 실제로 개선되면 최대 100만 Token 길이에서도 이점을 보였다.

Background - Transformer

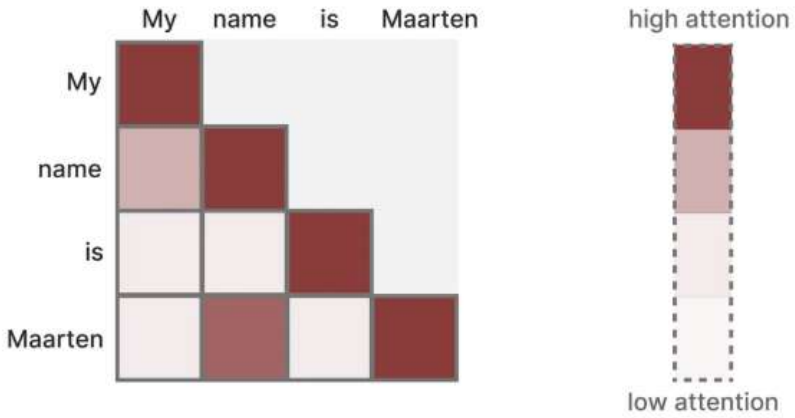


- 트랜스포머는 모든 입력을 Token으로 구성된 Sequence로 정의한다.

Ex) Hello! My name is 라는 문장이 있으면 Transformer는 이 문장을 각각의 Token으로 나누게 됨.

Background - Transformer

Transformer is capable of **selectively** and **individually** looking at **past tokens**.



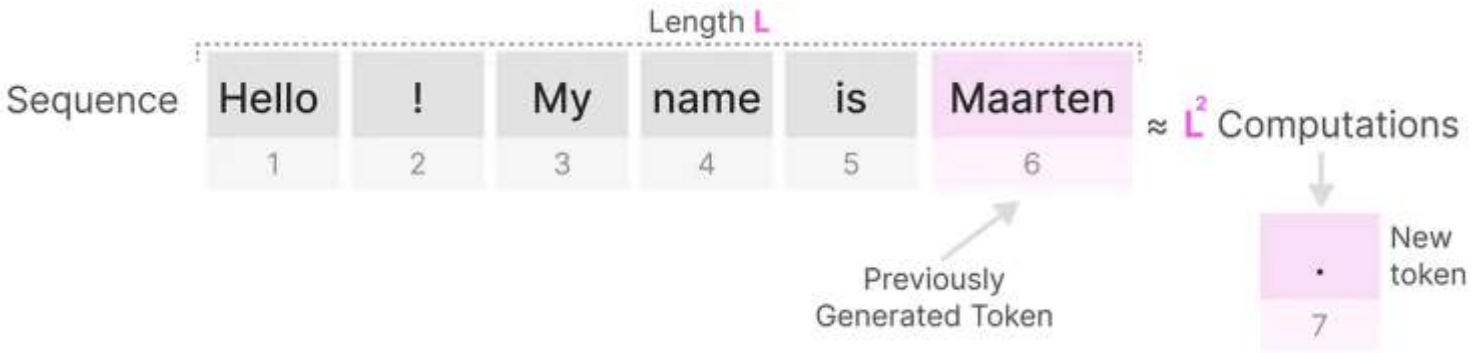
- 이러한 접근 방법은 Sequence 이전 Token과의 관계를 계산할 수 있게 해준다.
- 모든 Token info가 압축되지 않는 상태로 Attention이 계산된다.

Ex)학습 과정에서, 이 행렬은 한 번에 만들어진다. 'My'와 'name'간의 Attention을 계산하기 전에 구할 필요가 없다.

Self-Attention – 각 Token을 이전 Token과 비교하여 행렬을 만드는 데,

이 행렬의 가중치는 각 Token Pair에서 서로 간의 연관성으로 결정됨.

Background - Transformer

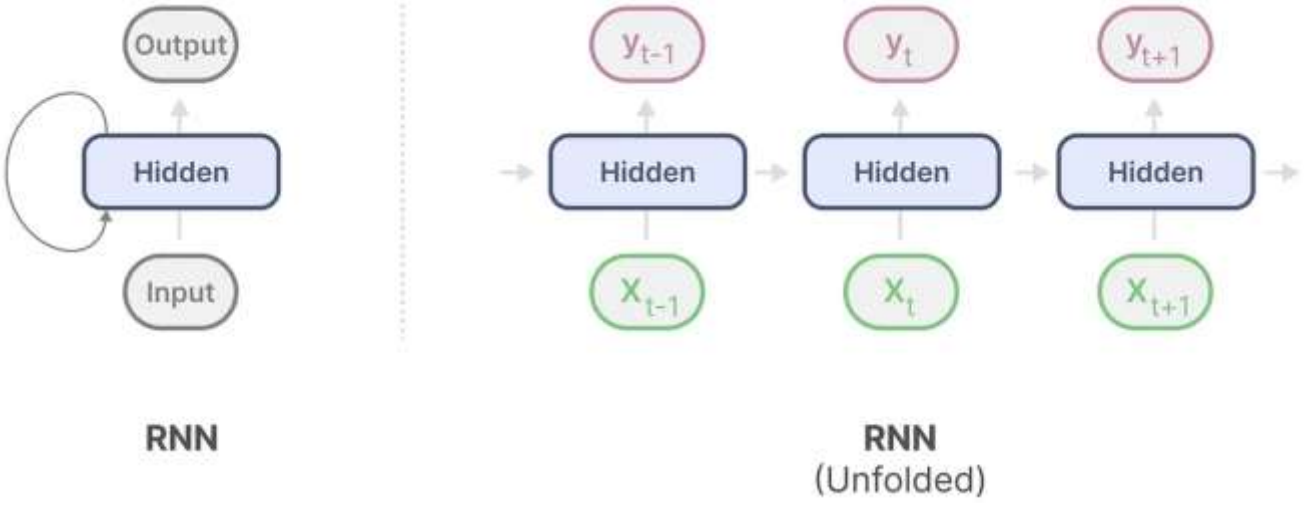


- But, next Token을 생성할 때, 다시 처음부터 전체 Sequence에 대한 Attention을 계산해야 한다.
- Transformer 알고리즘의 주요 Bottleneck.

Ex) If. GPT가 Hello! My name is Maarten 이라는 문장을 생성하고, 그 다음으로 next Token인 .을 생성할 때 다시 다 Self-Attention을 계산해야 한다.

Sequence가 늘어날 수록 계산해야 하는 Attention이 많아지므로, 계산 복잡도가 늘어남.

Background - RNN

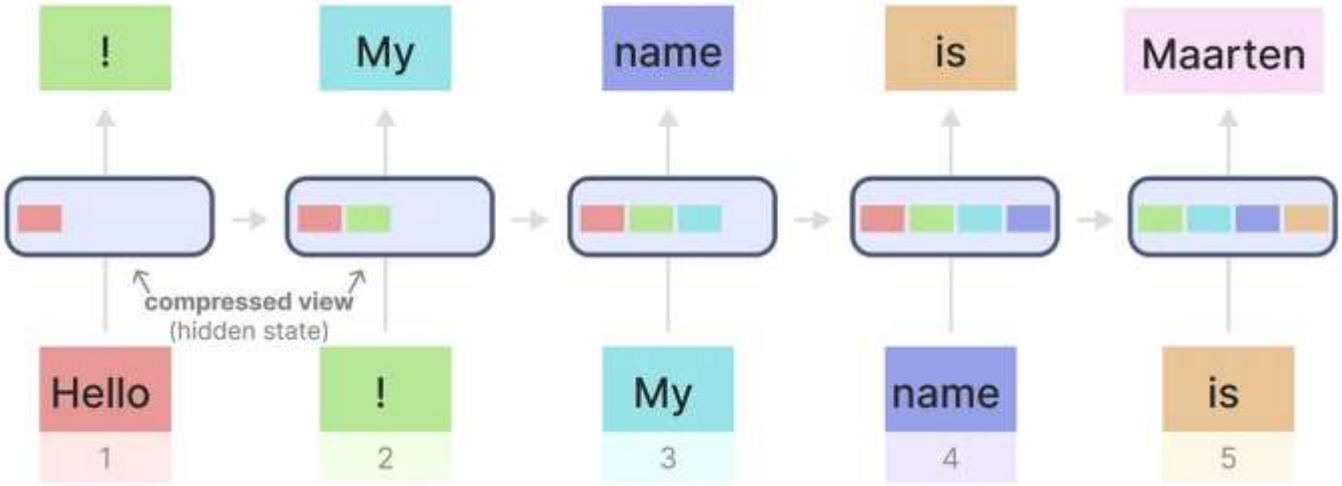


- RNN은 시간 t의 state와 이전 시간 t -1의 Hidden state 2가지 입력을 받아, 시간 t의 출력과 시간 t의 Hidden state를 생성.

Ex) input이 들어오면 이 안에서 Hidden State가 하나 만들어지고,

그 다음 Hidden State와 조합을 해서 다시 Hidden State를 만드는 형식

Background - RNN



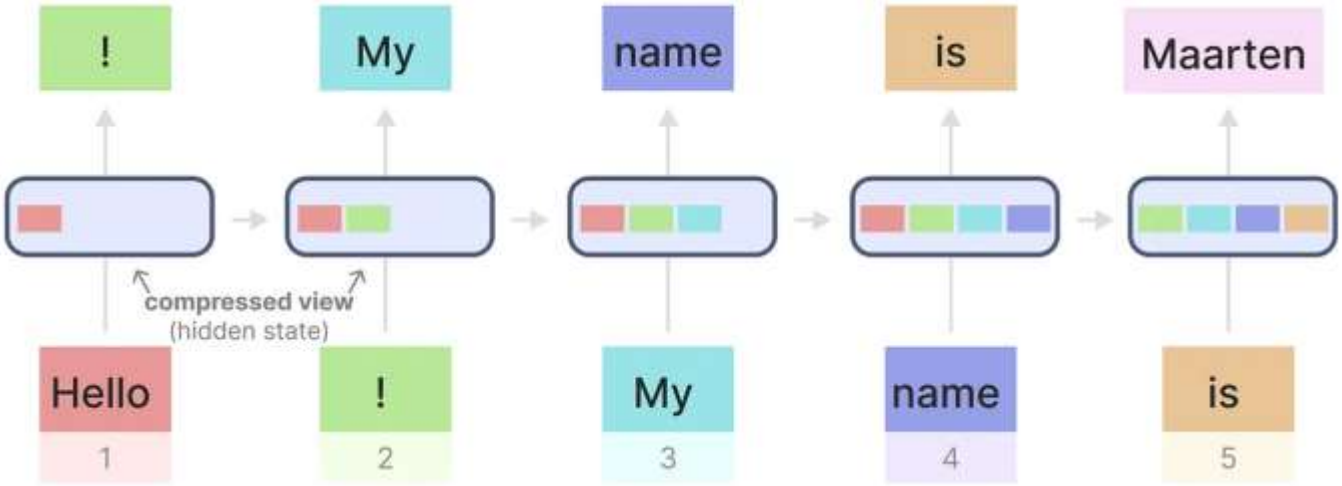
- RNN은 Transformer 처럼 모든 Token에 대한 Attention을 계산하지 않으므로 inference 속도가 매우 빠르다.

Ex) 이전 정보가 압축된 채로 들어오기 때문에 각각의 Attention을 구하지 않고 빠르게 생성할 수 있다.

But, 끝 부분에 왔을 때는 Hello 라는 info가 거의 손실됨.

RNN은 Transformer와 달리 빠르게 inference 할 수 있지만 각각의 Attention을 구하지 않기에 info가 손실될 수 있음.

Background - RNN

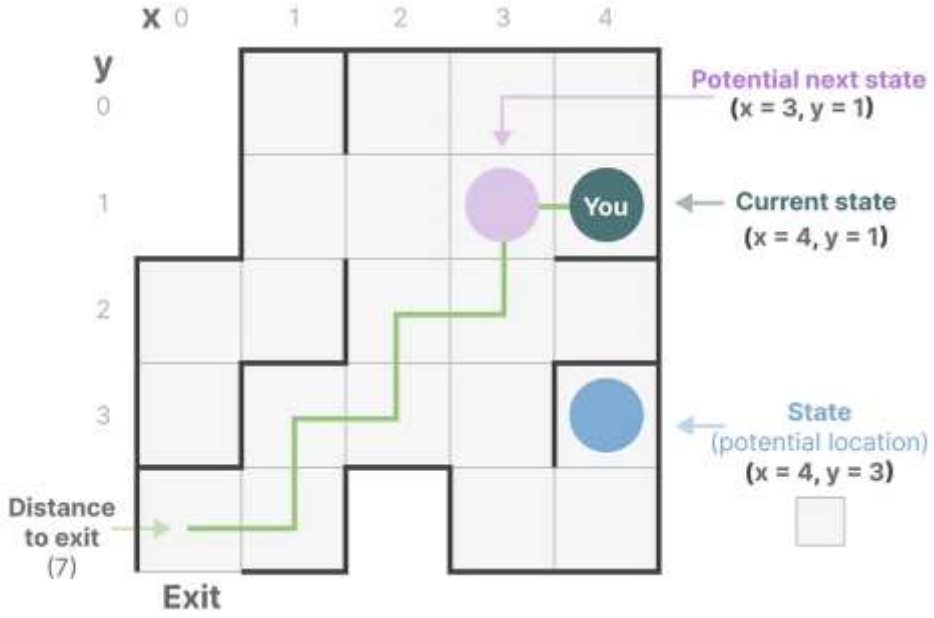


- But, Transformer와는 달리 이전 info들이 계속 압축되기 때문에 시간이 지속됨에 따라 처음 info들이 잊어지는 경향이 있다.
- 또한, RNN의 순차적인 특성 때문에 Training을 병렬적으로 수행할 수 없다.

Ex)Transformer 같은 경우는 가능한 Token들을 한꺼번에 Matrix 형태로 표현이 가능하기 때문에 병렬적 수행이 가능함.

But, RNN의 경우는 순차적으로 처리를 하기 때문에 병렬적 수행을 할 수 없다.

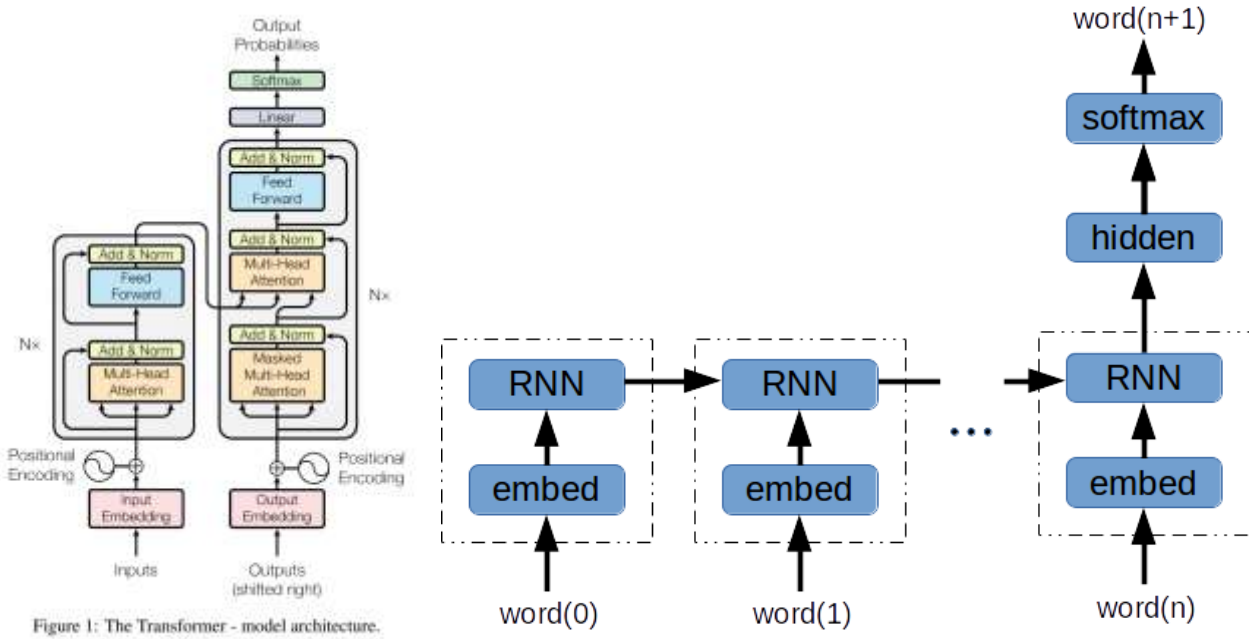
State Space Model (SSM)



- State Space : System을 완전히 설명하는 최소한의 변수로 정의된 공간.

Ex)if, 미로가 있을 때 2차원 Matrix로 표현, 하나의 State Space.

State Space Model (SSM)

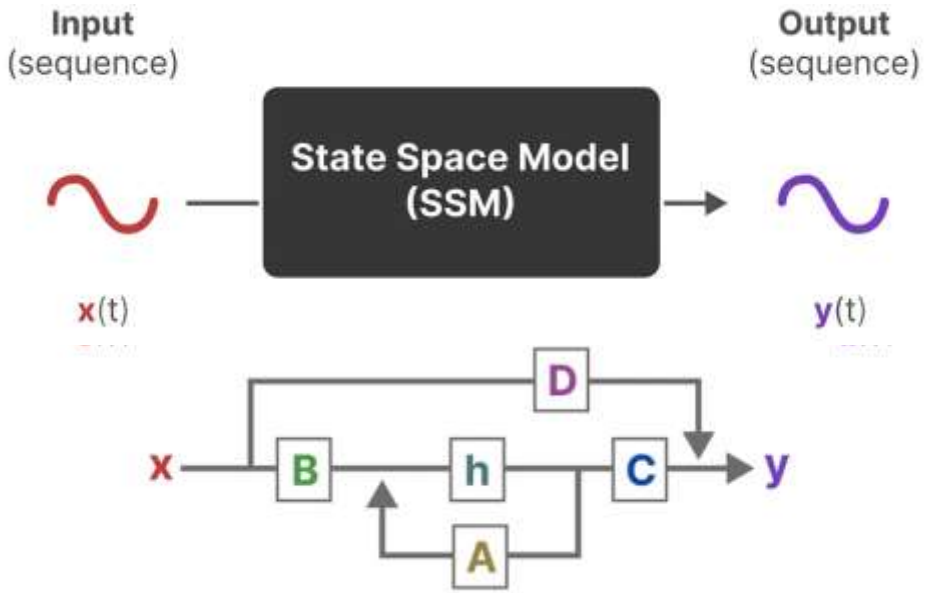


- State Space : System을 완전히 설명하는 최소한의 변수로 정의된 공간.

Ex)State Space가 AI와 무슨 상관인가. Transformer 같은 경우에는 Word가 처음 들어오게 되면 Embedding Processing을 하게 됨.

RNN의 경우에도 Embedding Processing을 하게 되는데, Embedding 자체가 Law한 Signal을 우리가 생각되는 Best한 Space로 Mapping 하는 것 자체가 state Space를 이미 하고 있다.

State Space Model (SSM)

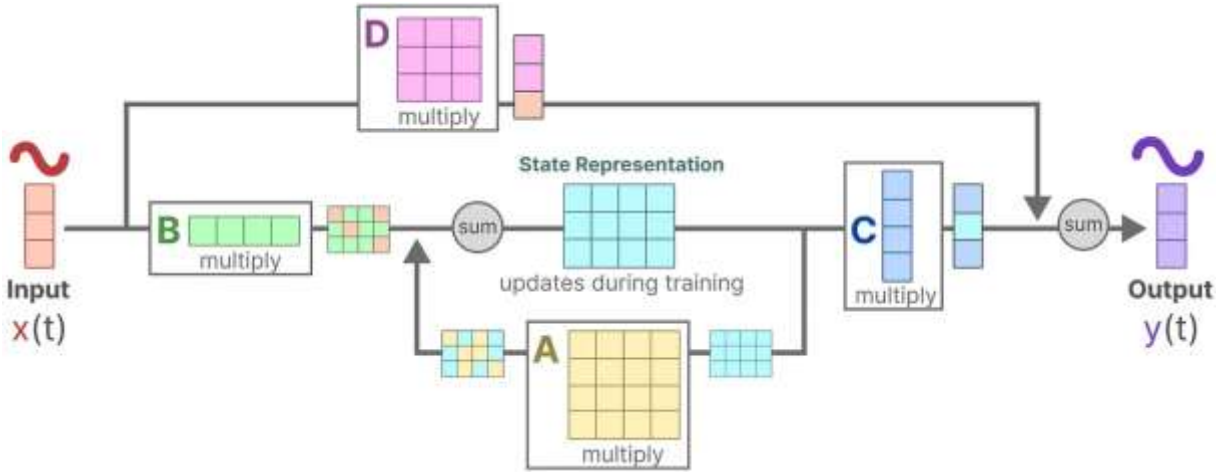


- State Space Model(SSM) : State Space 에서 정의된 입력 State에 따라 next State가 어떻게 될지 예측하는 Model.

Ex)if. This System, Input Signal이 System 안에 Output을 해당하는 Sequence Signal 있다고 가정.

입력되는 State가 next State 어떻게 되는 지 예측하는 수학적 모델.

State Space Model (SSM)



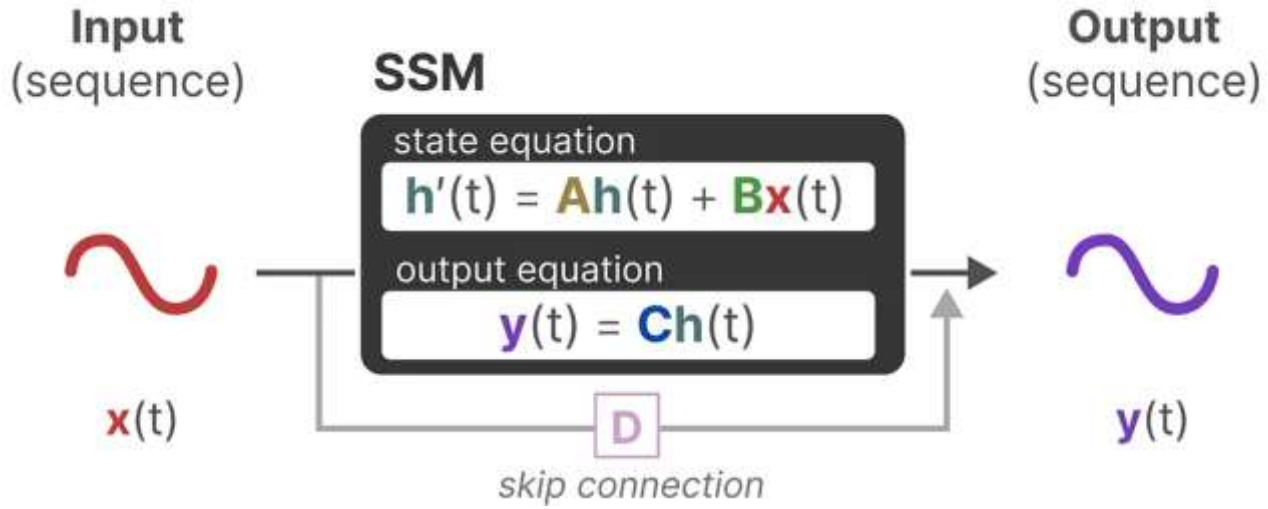
- A : t -1 시점 State에서 어떤 info를 가져갈 것 인가. System 내부에 시간에 흐름에 따른 State의 변화를 표현
- B : Input Signal을 State Space에 Mapping.
- C : System 내부의 State를 Signal info로 변환.

Ex)State Space Model은 4개의 Matrix로 구축을 해서 선형적으로 Stem을 서술한다.
먼저 B 같은 경우는 law한 signal을 우리의 state에 맞게 Embedding. 최초의 state. -> 시간에 따라 바뀌는 State info를 표현한 것이 A. ->

A + B : 이전 State와 new signal이 합쳐져 새로운 State를 만든다. ->

실제 환경에 맞는 info로 변환하기 위한 C -> D. System 외부에 있는 State를 서술하는 데, 별로 생각을 안 해도 됨.

State Space Model (SSM)

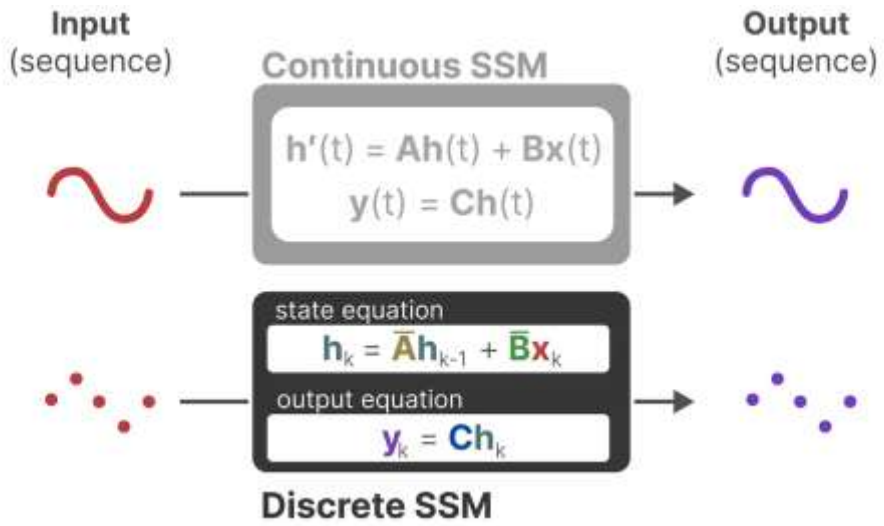


- A : t -1 시점 State에서 어떤 info를 가져갈 것 인가. System 내부에 시간에 흐름에 따른 State의 변화를 표현
- B : Input Signal을 State Space에 Mapping.
- C : System 내부의 State를 Signal info로 변환.

Ex)최초의 signal이 왔을 때 B로 우리가 정의한 Space로 mapping을 통해, if 이제 최초의 state가 없으면 Bx가 최초의 State가 되고, 있다면 A와 곱해진 A + B 의 식을 h라는 next State가 만들어지게 됨.

H가 C와 곱해져서 우리가 실제 Signal Space와 정의된 Mapping Process를 통해 설명할 수 있다.

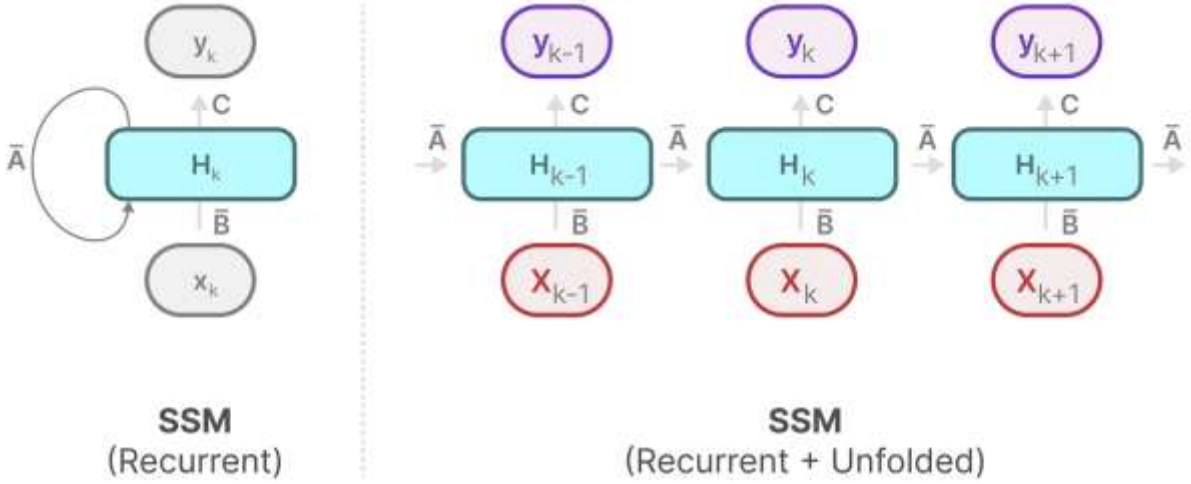
State Space Model (SSM)



- 우리는 실제 연속적인 값을 사용할 수 없기 때문에 입력 값과 출력 값을 이산화해야 한다.

Ex)컴퓨터는 이산적이기에 연속적인 값을 사용하기에는 어렵기에 이산화가 필요하다.

State Space Model (SSM)



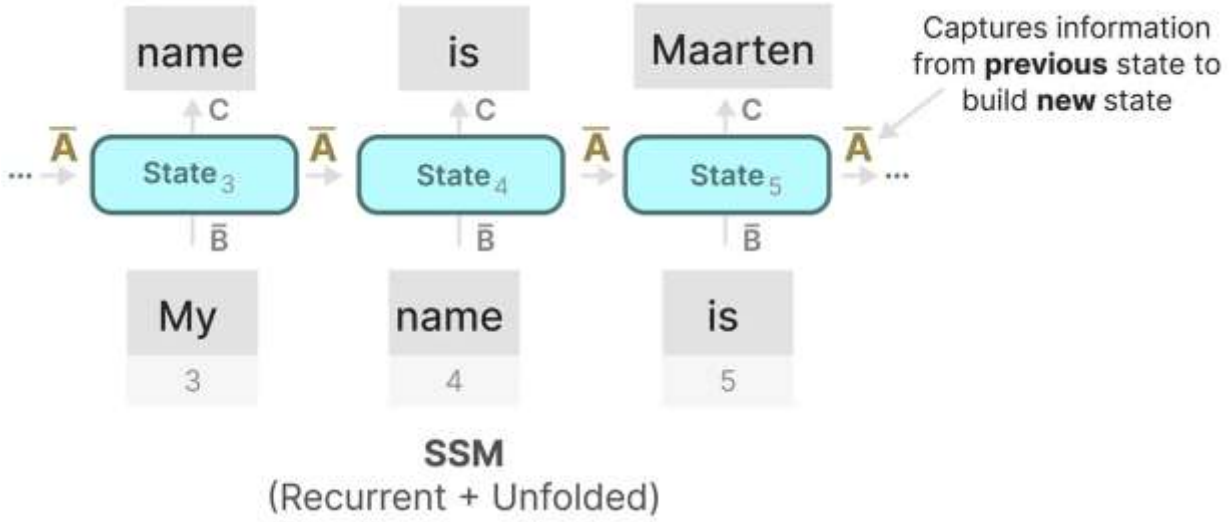
- State Space Model(SSM) >> RNN의 추상화

Ex) B Matrix. 실제 있는 data를 우리가 정의한 Space에 맞게 Embedding 함.

RNN도 똑같음. Embedding을 함. A의 역할이 이전 info에 대한 것을 얼마나 가지고 있을 지. 등에 대한 것이 RNN과 SSM이 비슷함.

저자는 sequence한 Model을 처리하기 위해서 SSM을 사용함.

Structured State Spaces for Sequences

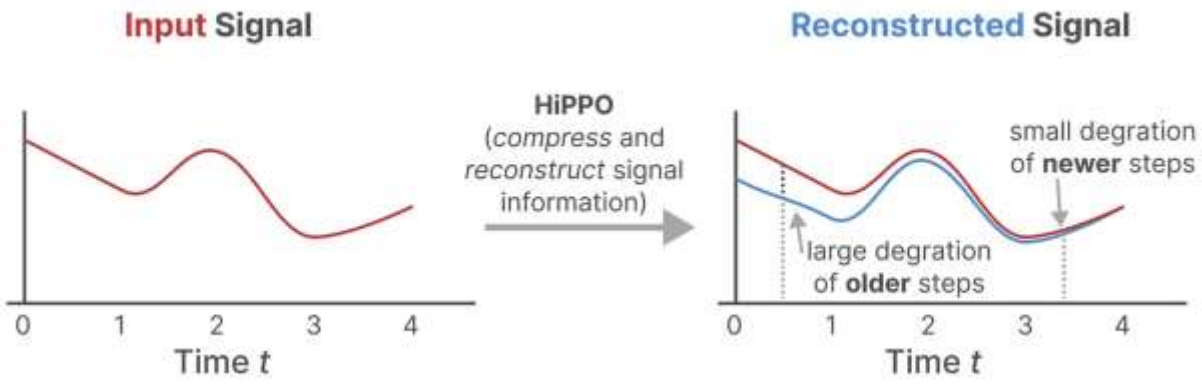


- Matrix A는 SSM 공식의 가장 중요한 역할을 한다.
- Matrix A는 이전 State에 대해서 계속 유지할 info를 추려내기 때문이다.

Ex) S4!, Mamba 이전에 Model. A Matrix 가 제일 중요하다.

A를 어떻게 설계할까?

Structured State Spaces for Sequences

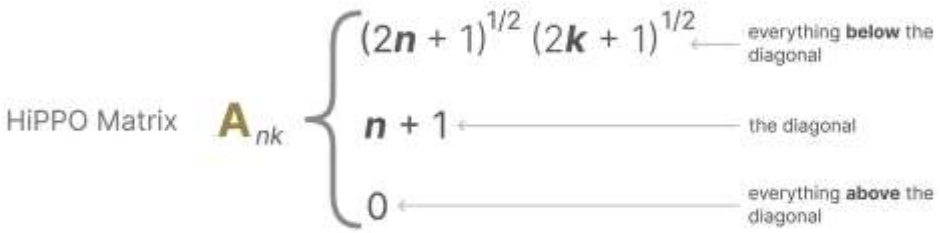


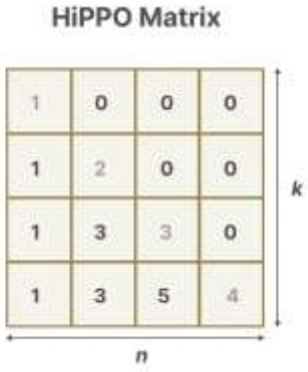
- 오래된 Signal (초기 Token)보다 새로운 Signal(최근 Token)에 더 큰 가중치를 두자!
- High-order Polynomial Projection Operators (NeurIPS 2020) 를 사용하여 Matrix 를 초기화.

Ex) Hippo를 사용한다. Matrix를 만드는 방식을 통해 A를 초기화하고 초기화된 Matrix를 RNN에 넣었을 때 잘 된다! 성능이 좋다!

최근에 있는 Sequence info는 매우 ACC가 좋고 old한 정보는 손실된 상태로 초기화가 됨.

Structured State Spaces for Sequences

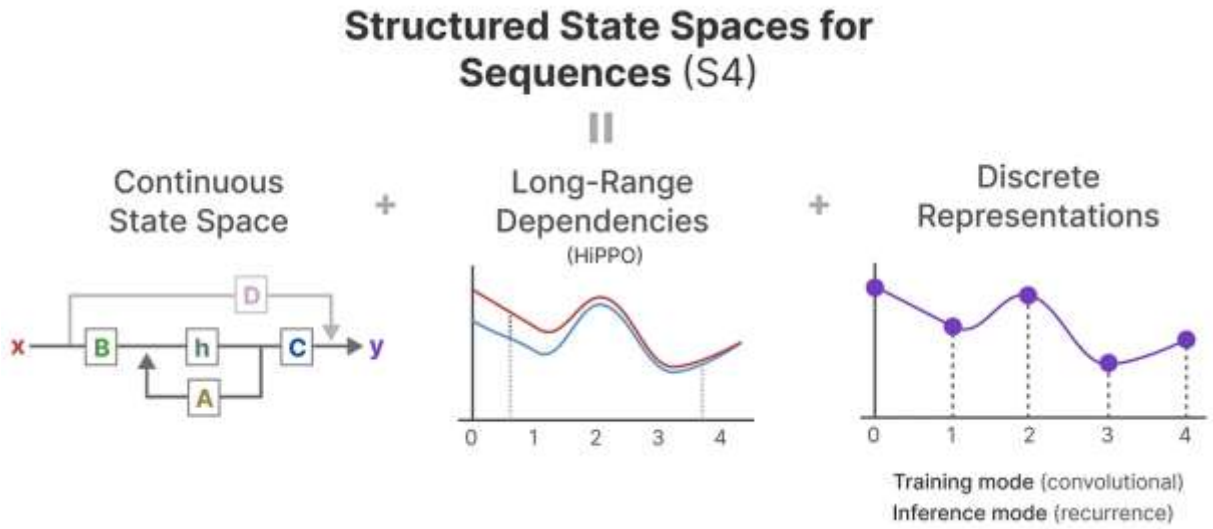




- High-order Polynomial Projection Operators (NeurIPS 2020) 를 사용하여 Matrix 를 초기화.
- 오래된 Signal (초기 Token)보다 새로운 Signal(최근 Token)에 더 큰 가중치를 두자!

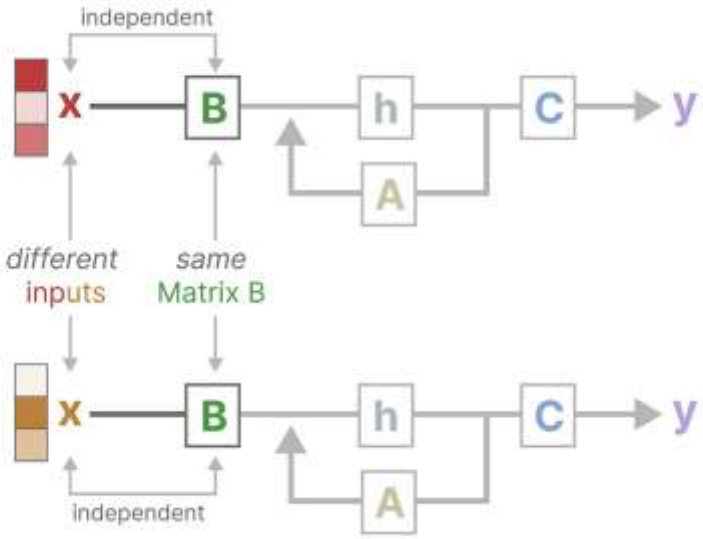
Ex) Matrix는 위 수식에 따라서 초기화가 되는 데, 보존이 잘 되더라.

Structured State Spaces for Sequences



- CSS + LRD(Hippo) + DR(이산화 과정)

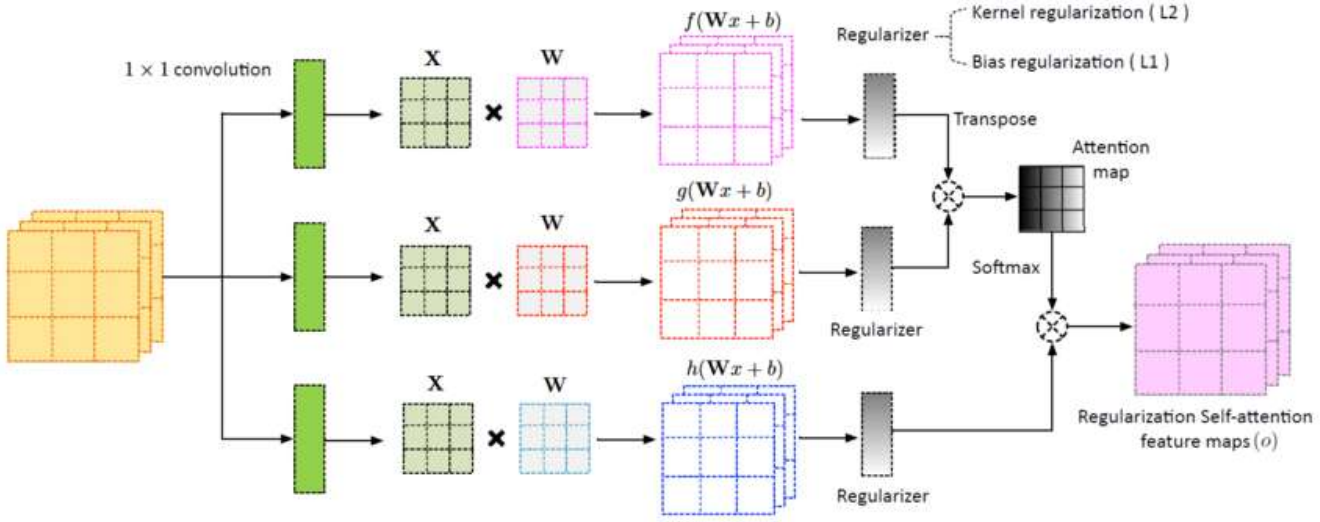
MAMBA – Selective SSM



- S4의 문제점은 A, B , C Matrix가 input에 상관없이 고정되어 있다.

Ex) 저자가 주장. Transformer 보다 좋을 것이 없다.

MAMBA – Selective SSM



- Transformer가 잘 작동하는 이유는 전체 Token에 대한 Attention을 계속 계산하는 것 뿐만 아니라 Input에 Dependent 한 Weight을 만들 수 있기 때문이다.

Ex) Attention 자체가 Weight가 있는 convolution 인데, 쿼리 키에 만들어진 input에 따라 Weight가 계속 바뀐다.

S4는 구현이 안 되어 있음.

MAMBA – Selective SSM

Algorithm 2 SSM + Selection (S6)

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

- Selective Selection
- B, C and Delta Matrix를 만드는 Projection function을 학습시킴으로써 Input에 종속되는 State Space Model을 만들 수 있다.

Ex) B와 C에 대해서는 input에 따라서 만드는 Projection 하는 Network를 하나 둬서 학습함.

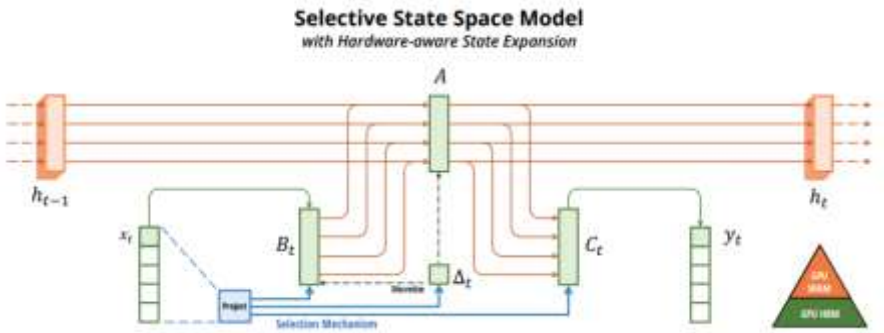
Why B and C? : A는 System 내부, B와 C는 real world에 있는 input을 다루기 때문에 학습을 통한 Network에 Projection으로 B, C를 구상함.

델타는 A, B에 이산화 과정을 통해 Tensor를 키우거나 작게 하는 것을 통해 어떤 info를 집중해서 훈련할 건지를 위해 추가.

MAMBA – Selective SSM

Algorithm 2 SSM + Selection (S6)

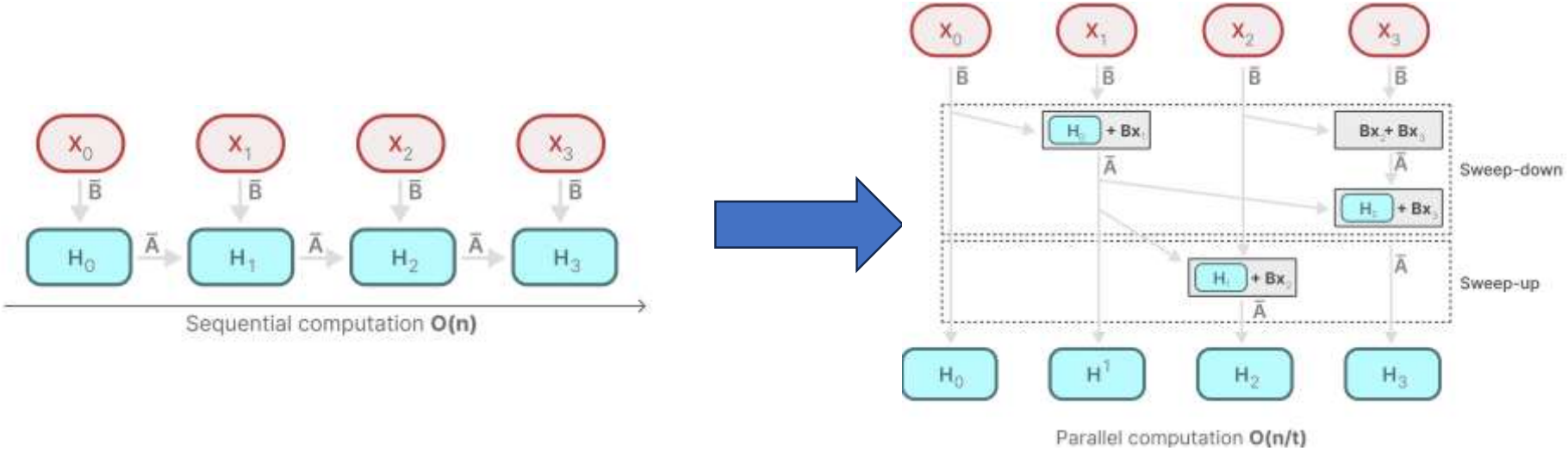
Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
1: $\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_B(x)$
3: $\boldsymbol{C}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



- Selective Selection
- B, C and Delta Matrix를 만드는 Projection function을 학습시킴으로써 Input에 종속되는 State Space Model을 만들 수 있다.

Ex) X input에 따라서 B, C Matrix와 델타를 구성한 후에 이것을 통해 Matrix가 변하는 Transformer와 비슷하게 수행하는 기능을 만들어 냄.

MAMBA – Selective SSM

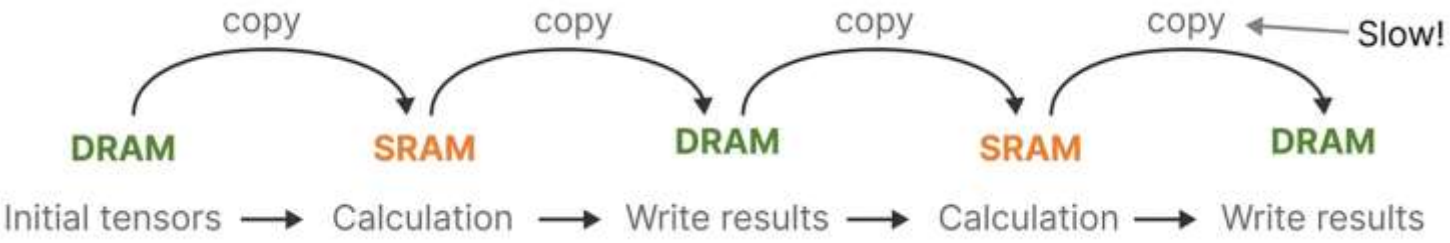


- The Scan Operation : RNN 형태에서도 병렬화가 가능하게 설계
- 계산할 수 있는 것은 먼저 계산해 놓고 있자!

Ex) Self – Attention X - RNN은 추론을 잘 한다.

RNN의 병렬화를 위해서 구조를 설계함.

MAMBA – Selective SSM

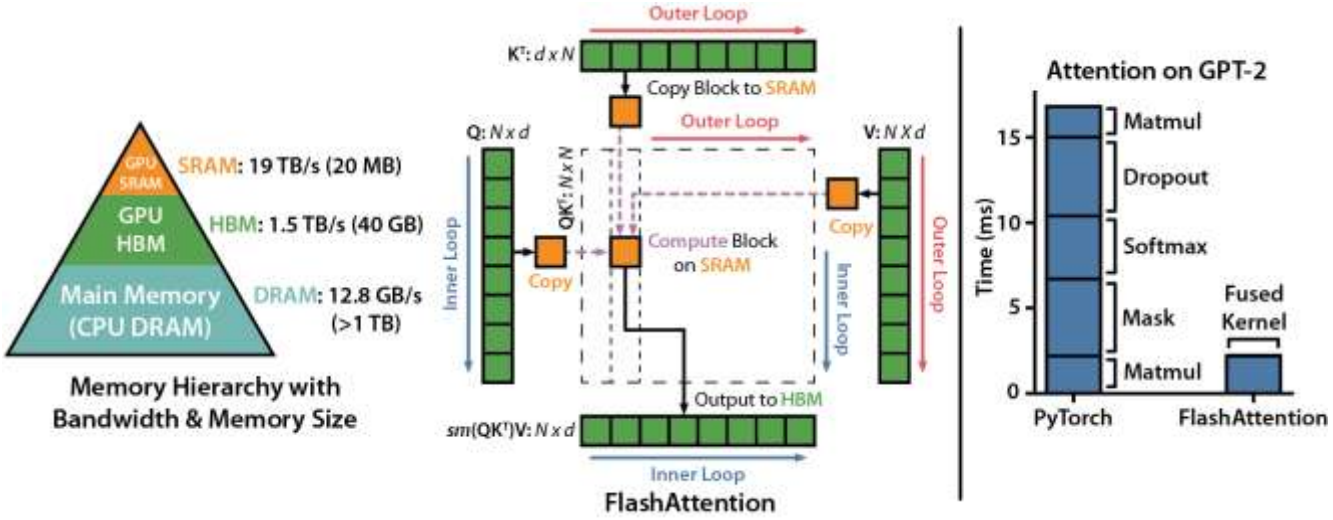


- Kernel Fusion : Hardware-aware Algorithm
- GPU의 주요 병목 현상은 SRAM과 DRAM 사이의 Copy and PASTE에서 발생.
- 저자는 이러한 memory IO 로 발생하는 병목 현상을 줄이기 위하여 Kernel fusion을 사용하였다.

Ex) Mamba를 Transformer보다 더 빠르게 하기 위해서 제안.

Transformer는 왜 느린가. RAM에서 서로 data를 옮기는 과정에서 병목 현상이 일어나 느리다!

MAMBA – Selective SSM



- Kernel Fusion : Hardware-aware Algorithm
- Idea : FlashAttention : Fast and Memory – Efficient Exact Attention with IO – Awareness.

Ex) 병목 현상을 어떻게 해결하냐!

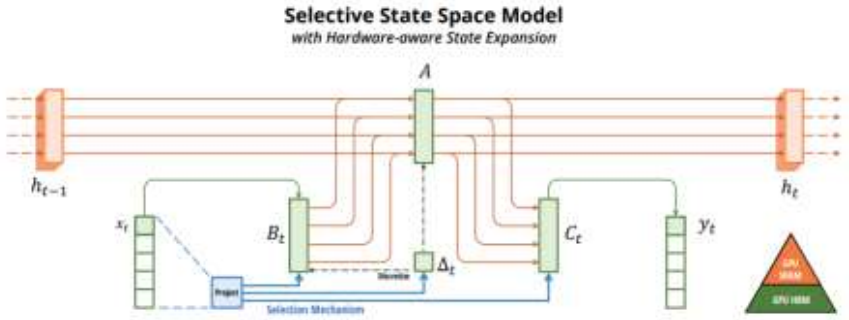
이전 연구 논문에 idea를 가지고 와 하드웨어의 연산의 순서를 고려함.

MAMBA – Selective SSM

Algorithm 2 SSM + Selection (S6)

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

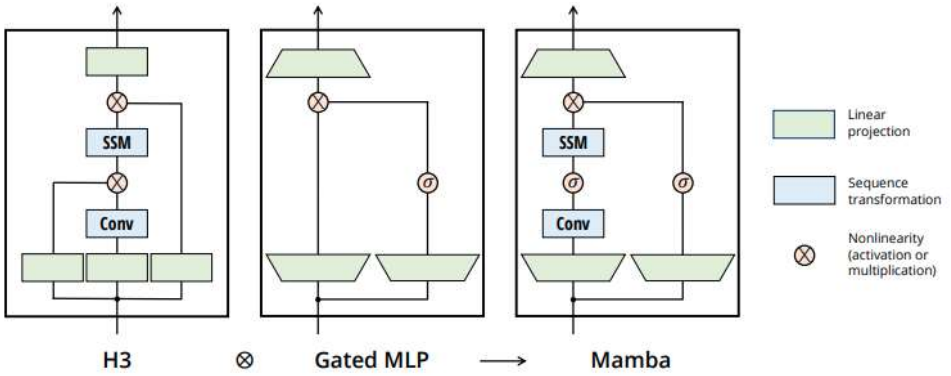
$\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
$\boldsymbol{B} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_B(x)$
$\boldsymbol{C} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_C(x)$
$\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D})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

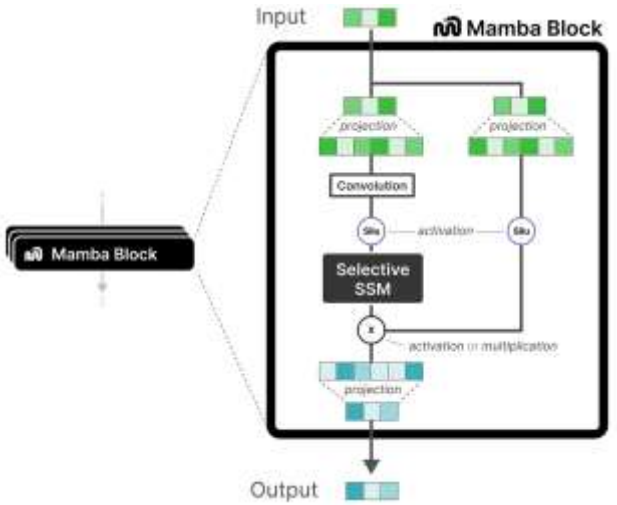


- Kernel Fusion : Hardware-aware Algorithm
- 3개의 Dimension을 가진 B, C, 델타를 HBM 에서 SRAM으로 옮긴다.
- Discretization(이산화)을 SRAM에서 바로 진행한다.
- SSM 후 결과물 y를 HBM으로 옮긴다.

Ex) 3개의 Tensor를 HBM -> SRAM 에서 이산화 작업. (주장.)

MAMBA – Selective SSM





- 최종적인 Mamba Architecture

Ex) 지금까지 봐온 게 Selective SSM 하나의 모듈.

H3 + Gated MLP -> Mamba

Empirical Evaluation

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**

- Selective Copying
- 기억해야 할 Token의 위치를 계속 바꾸면서 Langue 모델의 Attention 정도를 측정하기 위한 실험.
- MAMBA의 Architecture 보다는 Selective SSM이 Selective Copying에 중요한 역할을 한다는 것을 알 수 있다.

Ex) 평가. 저자는 언어 모델에서 중요한 것이 무엇인가.

Selective Copying -> 모든 명사를 뽑아라. -> 어디에 Attention을 두어야 하는 지 제대로 되지 않으면 output이 이상하게 나온다.

저자가 말하는 S6 99.8.

Empirical Evaluation

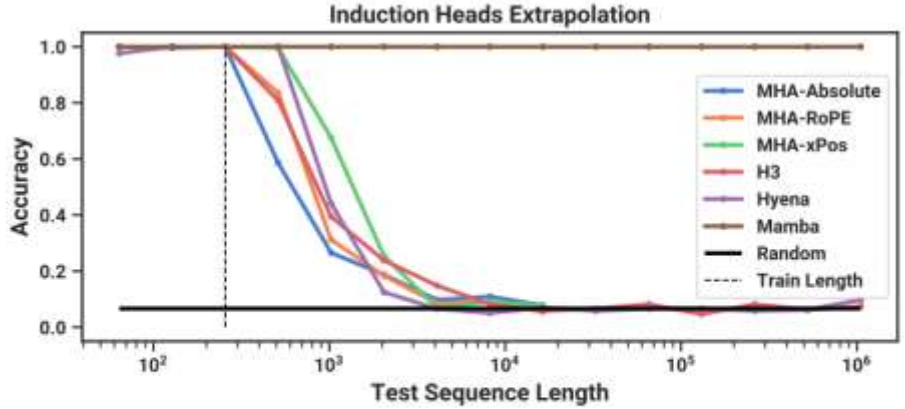
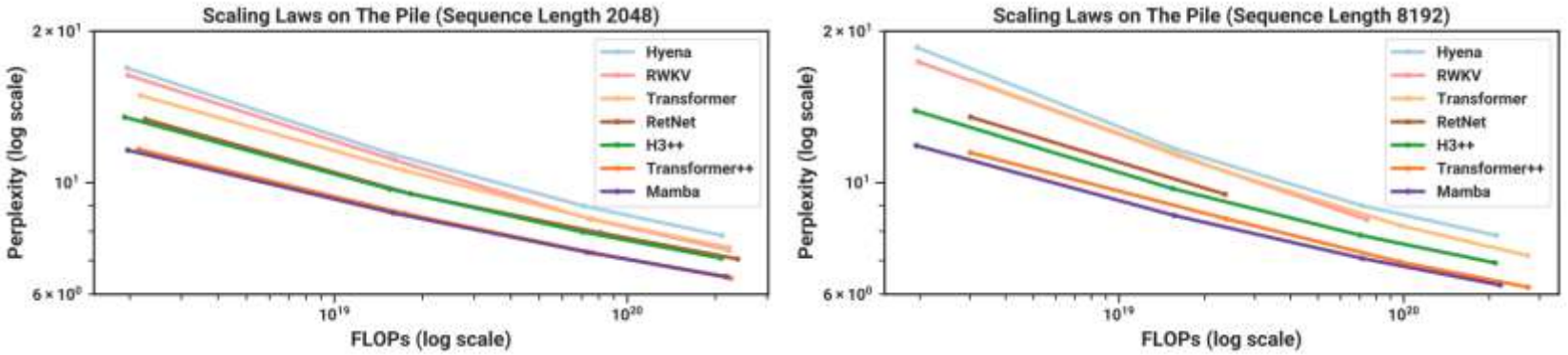


Table 2: (**Induction Heads.**) Models are trained on sequence length $2^8 = 256$, and tested on increasing sequence lengths of $2^6 = 64$ up to $2^{20} = 1048576$. Full numbers in Table 11.

- Induction heads
- 주어진 문맥을 바탕으로 해당 위치에서 답을 추론해야 하는 Task
- 이 또한 언어 모델의 Attention에 대한 성능을 평가할 수 있는 test.
- MAMBA의 경우 Sequence 길이가 길어지더라도 성능이 떨어지지 않음을 알 수 있다.

Ex) Induction heads. If 10^2.5까지 훈련을 한다고 했을 때, 어느 정도의 Sequence 길이의 위치에서 들어가야 할 가장 적절한 단어를 훈련한 곳에서 찾아라. T가 길어져도 잘 학습한다!!. A,B, 델타 덕분.

Empirical Evaluation



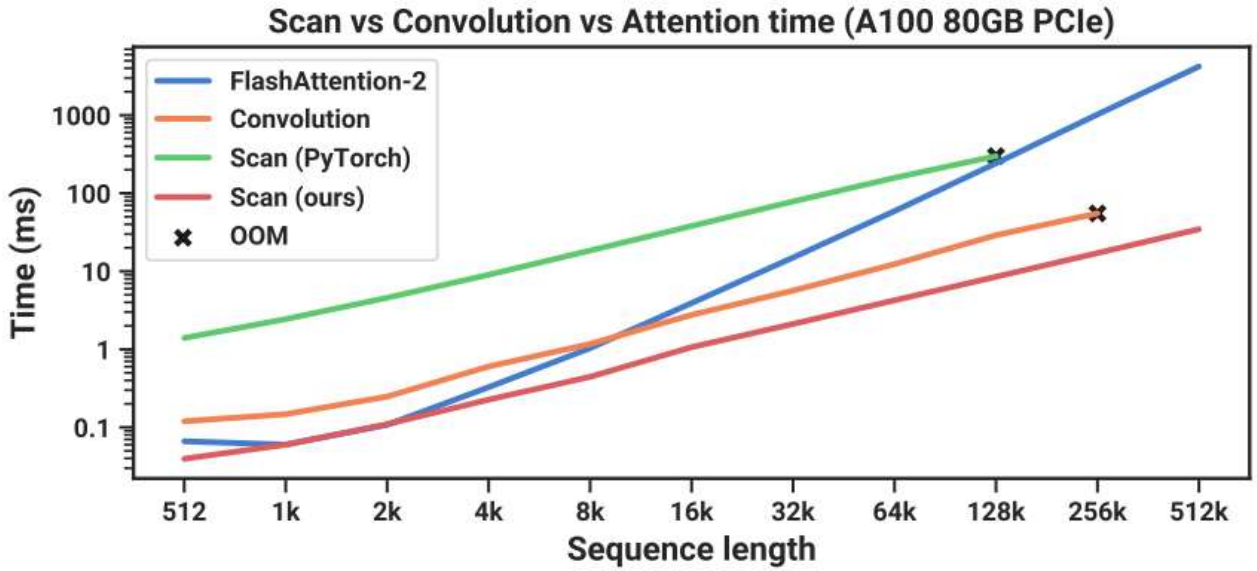
- Scaling Law
- 모델의 크기가 커질수록 성능 또한 좋아지는 지를 평가.
- FLOPS(Floating point OperationS) : 연산량 횟수.
- Perplexity : 언어 모델이 예측한 다음 단어의 확률의 역수, '혼란 정도' 를 나타냄.
- MAMBA가 Scalable 하다는 것을 보여줌.

Ex) 언어 모델은 최대한 키움으로 써 성능을 높이는 방향

예측을 못할 수록 Perplexity를 높아짐. 낮을수록 좋은 거죠.

모델 크기가 높아질수록 좋은 결과를 보이는 것을 볼 때, 자신들의 모델이 Large 언어 모델에 적용될 수 있다는 것을 보여줌.

Empirical Evaluation



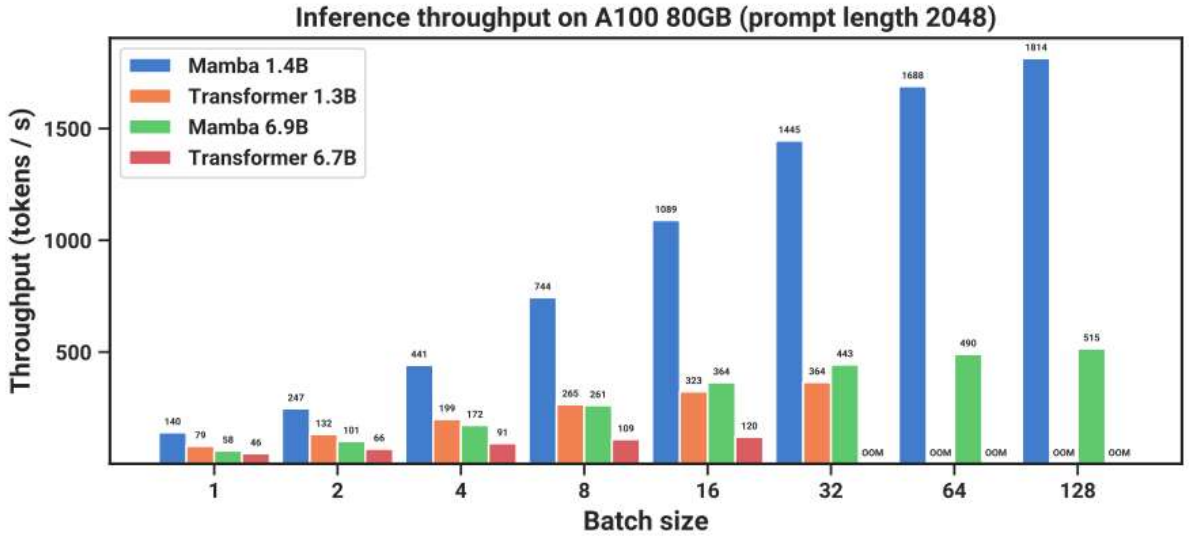
- Speed and Memory Benchmarks
- Sequence Length에 따른 Training time을 나타내는 Table.
- Scan (Ours) : JAX로 구현
- RNN에서 병렬이 가능한 Scan 구조로 바꾸었을 때 2k를 기점으로 훈련 시간이 Attention 이나 Convolution 보다 빠름을 알 수 있다.

Ex) 저자는 속도도 중요시 했음.

원래 코드 JAX라는 코드로 작성되어 있는 데, pyTorch를 돌려봄.

기존에 있는 Attention보다 빠른 속도로 훈련을 할 수 있음.

Empirical Evaluation



- Speed and Memory Benchmarks
- 같은 Parameter 숫자에 대해서 MAMBA가 Transformer 보다 inference 속도가 빠름을 알 수 있다.

Ex)1.4B 1.3B, 같은 Parameter 기준에서도 좋은 성능을 낼 수 있다.

Empirical Evaluation

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

- Downstream task – DNA Modeling.

Ex) 자신들의 모델이 여러 task에 있는 Backbone Model로 잘 활용될 수 있다.

Empirical Evaluation

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

- Ablation : Selective Parameters
- Discretize, B, C를 모두 Input Dependent 하게 설계하는 것이 가장 좋음을 알 수 있다.

Ex) 델타, B, C를 학습으로 만들었을 때 잘 학습이 잘 하냐.

Empirical Evaluation

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

- Ablation : Parameterization of A
- Random 하게 Matrix A를 초기화하는 것이 가장 좋음을 알 수 있다.

Ex) 저자는 normal Discretization 해서 하는 것이 가장 좋다!

Empirical Evaluation

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

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

- Ablation : SSM state dimension

Ex) dimension N 을 얼마나 주는 게 좋나.

각 B,C 16가지의 state dimension이 가장 좋다.

Embedding을 하게 되면 Token 마다 벡터를 늘리게 되는 데 그 State라고 생각하면 됨.

Conclusion

- 저자는 Self-Attention을 구하는 것이 아니라. RNN 계열에서 Attention 기반을 채용해 선형적이면서도 Transformer랑 비슷하거나 우위에 있을 수 있다.
- Selective-Space Model 기반으로 하는 기초 Model 구축에 대한 폭 넓은 응용 가능성에 큰 기대를 가지고 있으며,
- MAMBA가 일반적인 Sequence Model Backbone으로 활용될 수 있는 강력한 후보임을 시사하고 있다.

Sources

- https://modulabs.co.kr/blog/introducing-mamba
- https://arxiv.org/abs/2312.00752
- https://minyoungxi.tistory.com/118
- https://www.youtube.com/watch?v=JjxBNBzDbNk DSBA LAB
- https://www.youtube.com/watch?v=l-dQCTv9wIg AIR LAB
- https://github.com/state-spaces/mamba
- https://tulip-phalange-a1e.notion.site/05f977226a0e44c6b35ed9bfe0076839
  원문 저자: Maarten Grootendorst, 번역: 신종훈

THANK YOU

[2025.09.25. Journal Club]

Computer Science and Engineering
Incheon National University
Visual Information Perception Laboratory MINJE JEON