A Visual Guide to Mamba and State Space Models (SSM)

An Alternative to Transformers for Language Modeling



MAARTEN GROOTENDORST

FEB 19, 2024

https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Artificial Intelligence

Creating the Future

Dong-A University

Division of Computer Engineering & Artificial Intelligence

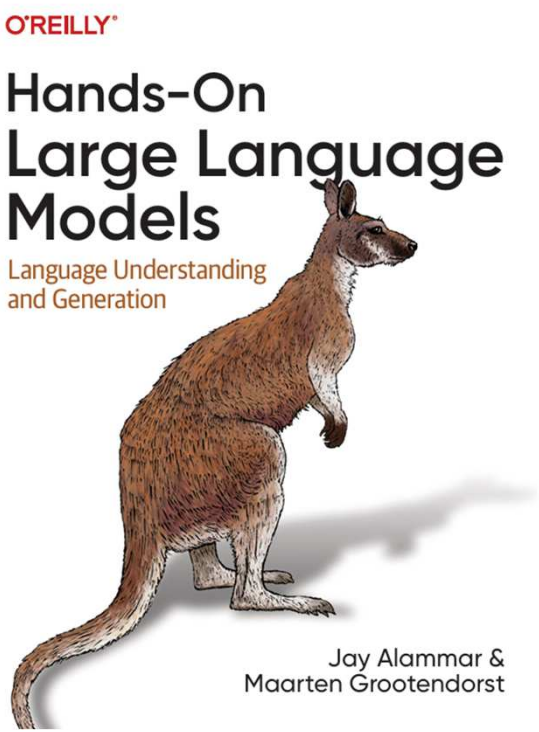
References

Exploring Language Models

- https://newsletter.maartengrootendorst.com/

Main

- https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state



Mamba

- Github : https://github.com/state-spaces/mamba
- Paper : https://arxiv.org/abs/2312.00752

Suggest Following Resources

- **The Annotated S4** is a JAX implementation and guide through the S4 model and is highly advised! (https://srush.github.io/annotated-s4/)
- A great YouTube video introducing Mamba by building it up through foundational papers. (https://www.youtube.com/watch?v=ouF-H35atOY)
- The Mamba repository with checkpoints on Hugging Face. (https://github.com/state-spaces/mamba, https://huggingface.co/state-spaces)
- An amazing series of blog posts (1, 2, 3) that introduces the S4 model. (https://hazyresearch.stanford.edu/blog/2022-01-14-s4-1, https://hazyresearch.stanford.edu/blog/2022-01-14-s4-2, https://hazyresearch.stanford.edu/blog/2022-01-14-s4-3)
- The Mamba No. 5 (A Little Bit Of...) blog post is a great next step to dive into more technical details about Mamba but still from an amazingly intuitive perspective. (https://jameschen.io/jekyll/update/2024/02/12/mamba.html)
- And of course, the Mamba paper! It was even used for DNA modeling and speech generation. (https://arxiv.org/abs/2312.00752)

References

Introduction of State Space Models (SSM)

- https://huggingface.co/blog/lbourdois/get-on-the-ssm-train

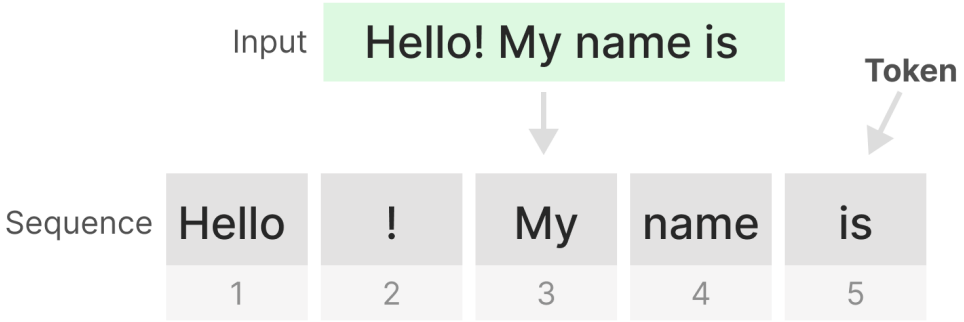
Part 1: The Problem with Transformers

➢ Transformer

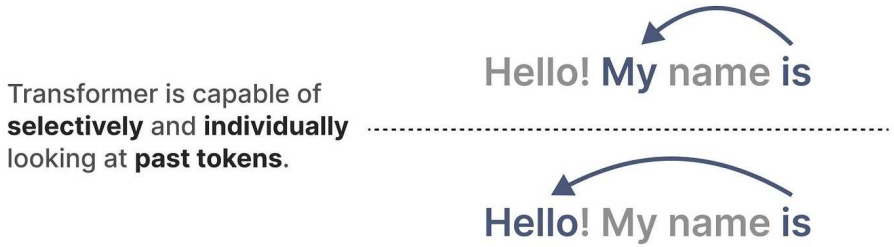
- To further improve LLMs, new architectures are developed that might even **outperform the Transformer architecture**. One of these methods is **Mamba**, a **State Space Model.**

➢ Short re-cap of transformers first and explore one of its disadvantages.

- A Transformer sees any textual input as a *sequence* that consists of *tokens*.

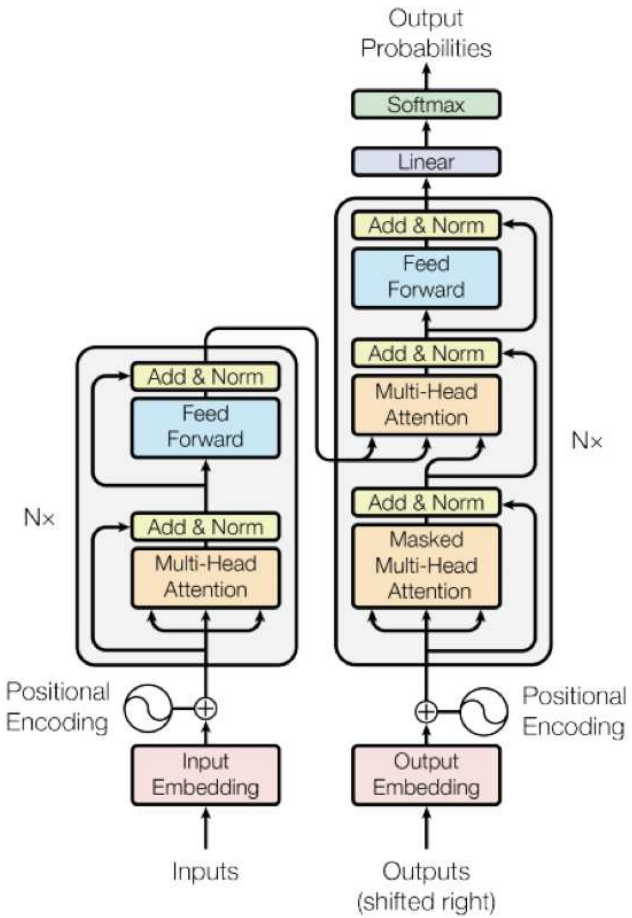


- A major benefit of Transformers : Whatever input it receives, it can look back at any of the earlier tokens in the sequence to derive its representation.

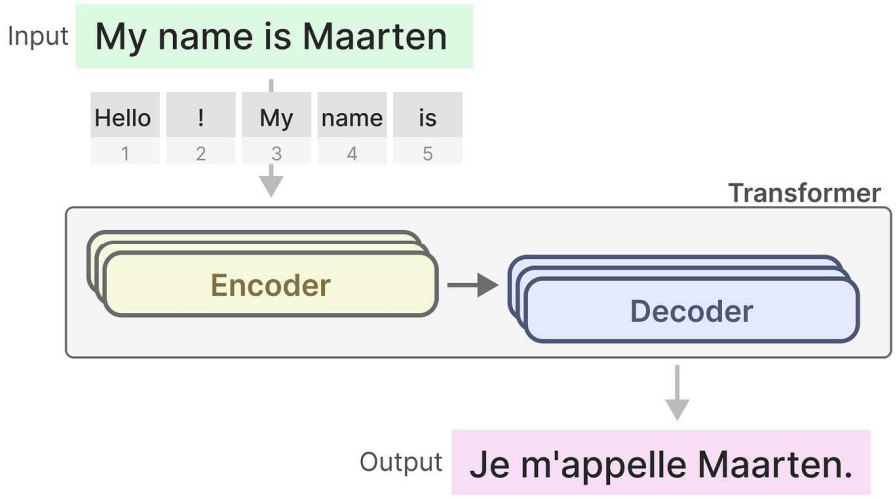


Part 1: The Problem with Transformers

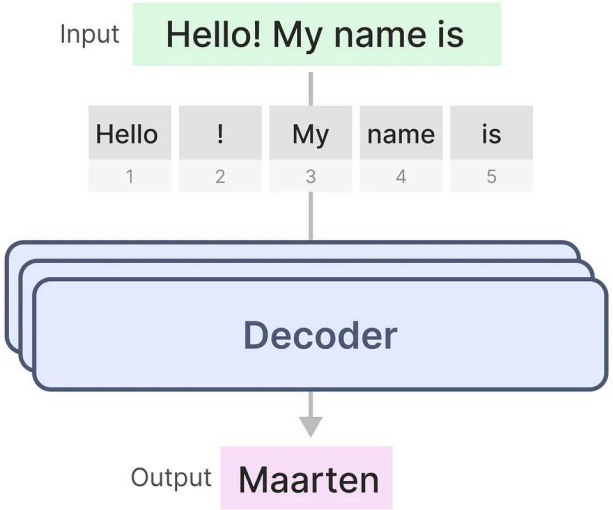
➢ The Core Components of Transformers



- Transformer consists of two structures, *a set of **encoder** blocks for **representing text*** and *a set of **decoder** blocks for **generating text***.



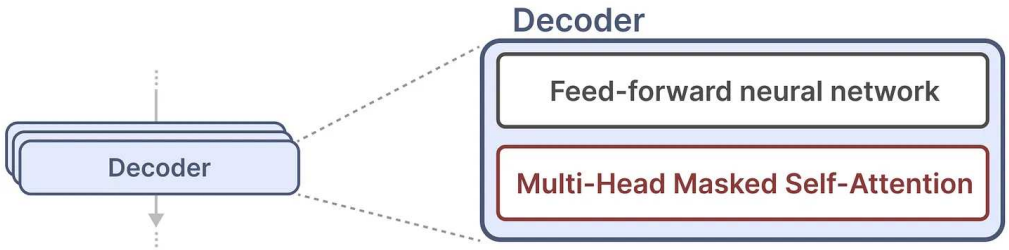
- We can adopt this structure to create generative models by using only ***decoders***. This Transformer-based model, *Generative Pre-trained Transformers (GPT)*, uses decoder blocks to complete some input text.

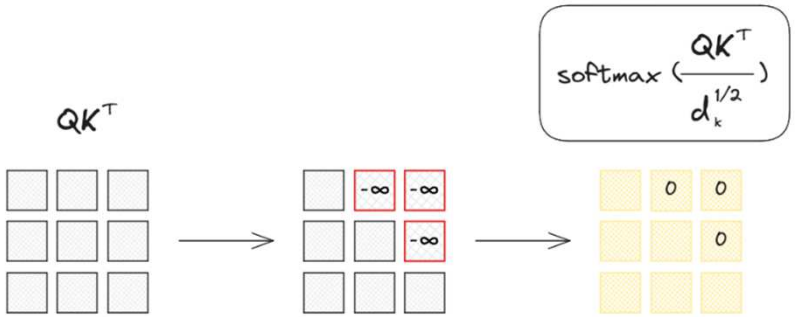


Part 1: The Problem with Transformers

➢ A Blessing with Training…

- A single ***decoder*** block consists of two main components, **masked self-attention** followed by **a feed-forward neural network**.

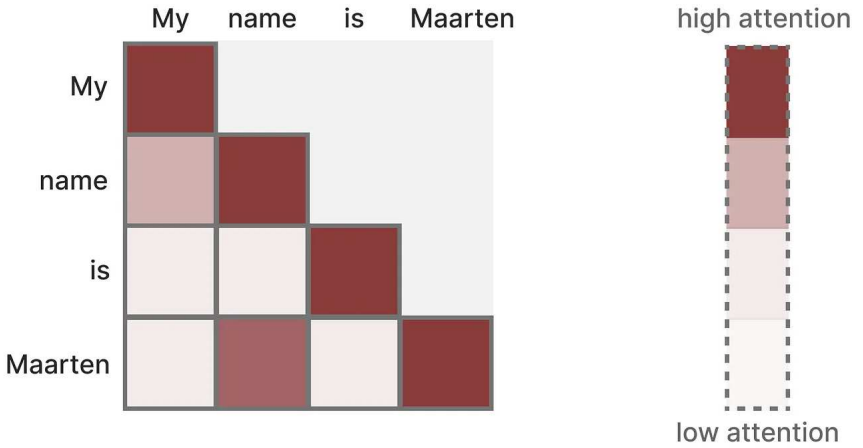




A **mask** is applied to **the attention scores to block connections to future tokens**. Tokens after the current position are assigned very large negative values (e.g., −∞) in the attention score matrix, making their contributions effectively zero after the softmax operation.

- **Self-attention** is a major reason why these models work so well. It enables an uncompressed view of the entire sequence with fast training.

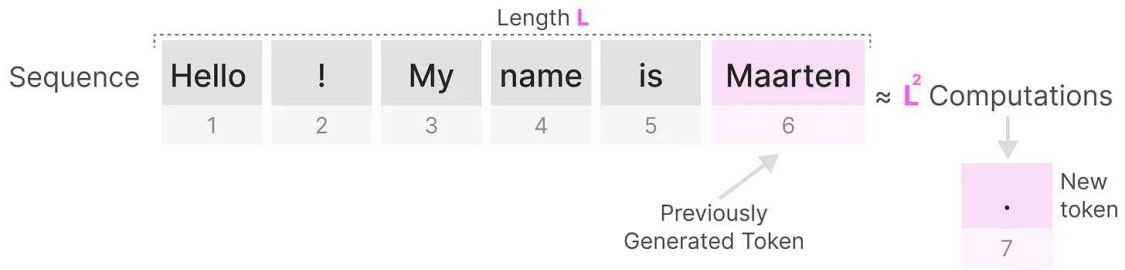
- So how does it work?
- It creates a matrix comparing each token with every token that came before. The weights in the matrix are determined by how relevant the token pairs are to one another.



Part 1: The Problem with Transformers

➢ **And the Curse with Inference!**

- **<u>When generating the next token, we need to re-calculate the attention for *the entire sequence*</u>, even if we already generated some tokens.**



	Training	Inference
Transformers	**Fast!** (parallelizable)	**Slow...** (scales **quadratically** with sequence length)

- <u>Generating tokens for a sequence of length L</u> needs <u>roughly **L² computations**</u> which can be costly if the sequence length increases.

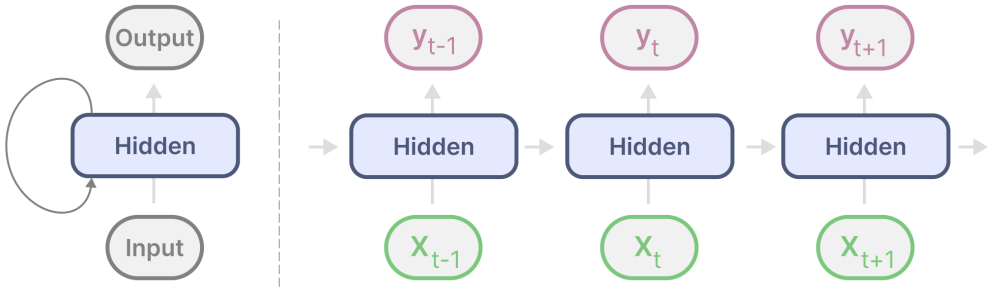
- This need to *recalculate the entire sequence* is a major bottleneck of the Transformer architecture.

- Let's look at how a "classic" technique, Recurrent Neural Networks, solves this problem of **slow inference**.

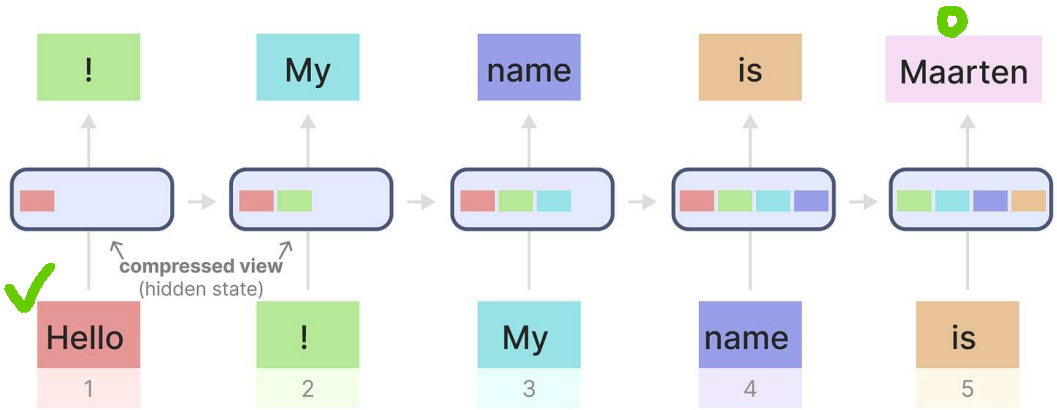
Part 1: The Problem with Transformers

➢ Are RNNs a Solution?

- Recurrent Neural Networks (RNN) is a **sequence-based network.** It takes **two inputs at each time step in a sequence**, namely the input at time step *t* and a hidden state of the previous time step *t*-1, to generate the next hidden state and predict the output.
- RNNs have a *looping mechanism* that allows them to pass information from a previous step to the next. We can "unfold" this visualization to make it more explicit.



- When generating the output, the RNN only needs to consider the previous hidden state and current input. It prevents recalculating all previous hidden states which is what a Transformer would do.
- In other words, RNNs can do inference fast as it scales linearly with the sequence length! In theory, it can even have an infinite context length.
- To illustrate, let's apply the RNN to the input text we have used before.



- Each hidden state is the aggregation of all previous hidden states and is typically a compressed view.

Part 1: The Problem with Transformers

➢ Are RNNs a Solution?

- There is a problem, however…
- Notice that the last hidden state, when producing the name “Maarten” does not contain information about the word “Hello” anymore. **RNNs tend to forget information over time since they only consider one previous state**.
- Although RNNs could be fast for both training and inference, they lacked the accuracy that the Transformer models could offer.
- Instead, we look at **State Space Models** to efficiently use RNNs (and sometimes use convolutions).

Part 2: The State Space Model (SSM)

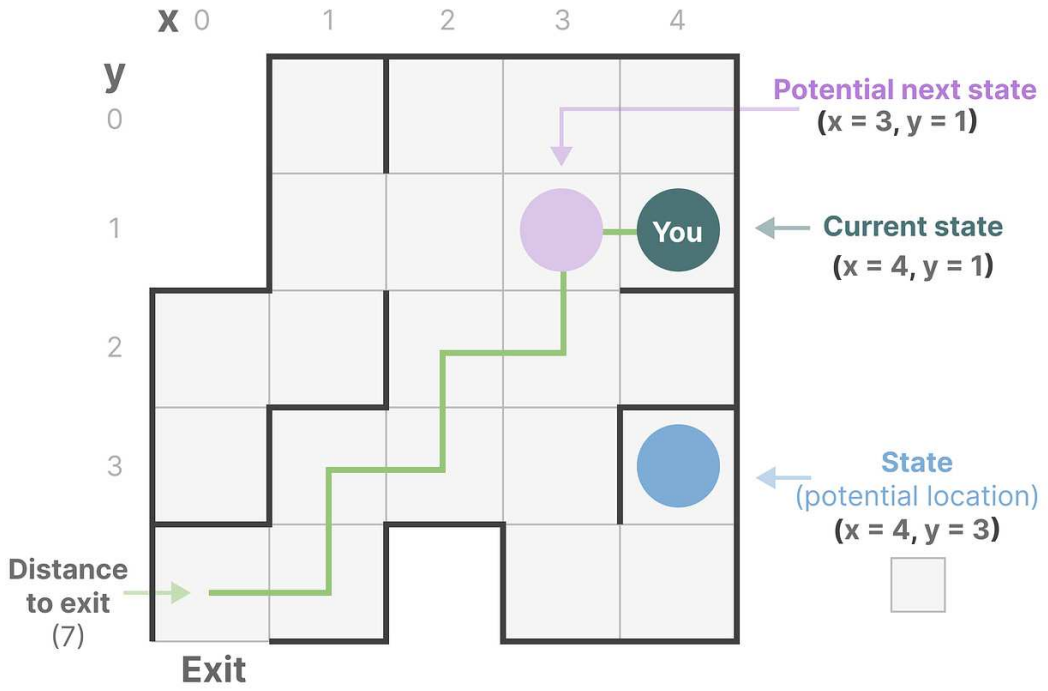
☞ The Annotated S4, “Efficiently Modeling Long Sequences with Structured State Spaces” https://srush.github.io/annotated-s4/

☞ Introduction to State Space Models (SSM), https://huggingface.co/blog/lbourdois/get-on-the-ssm-train

- A **State Space Model (SSM)**, like the Transformer and RNN, processes *sequences of information*, like text but also signals.

➢ What is a State Space?

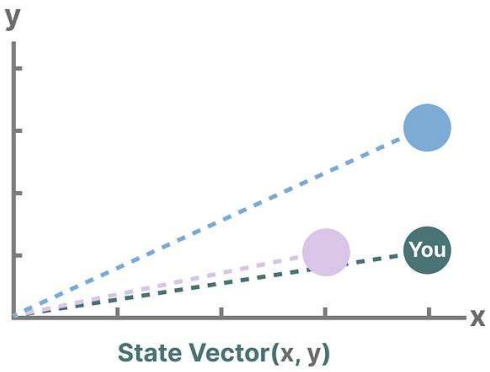
- A ***State Space*** contains the minimum number of variables that fully describe a system. It is a way to mathematically represent a problem by defining a system's possible states.
- Let’s simplify this a bit. Imagine we are navigating through a maze. The “***state space***” is the map of all possible locations (states). Each point represents a unique position in the maze with specific details, like how far you are from the exit.
- The “***state space representation***” is a simplified description of this map. It shows where you are (current state), where you can go next (possible future states), and what changes take you to the next state (going right or left).

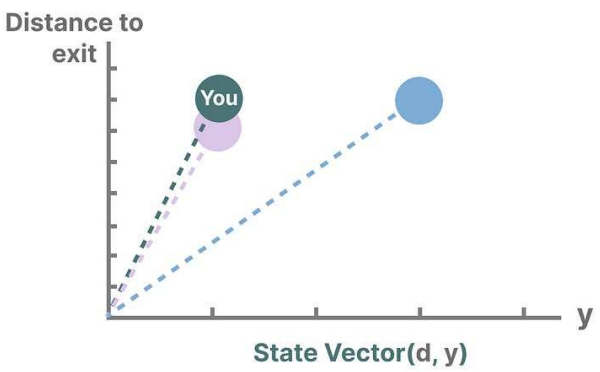


Part 2: The State Space Model (SSM)

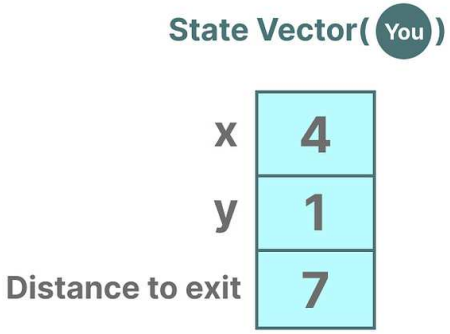
➢ What is a State Space?

- Although State Space Models use equations and matrices to track this behavior, it is simply a way to track where you are, where you can go, and how you can get there.
- The variables that describe a state, in our example the X and Y coordinates, as well as the distance to the exit, can be represented as "***state vectors***".





- Sounds familiar? That is because embeddings or vectors in language models are also frequently used to describe the "*state*" of an input sequence. For instance, a vector of your current position (state vector) could look a bit like this:.



- In terms of neural networks, the "***state***" of a system is typically its ***hidden state*** and in the context of Large Language Models, one of the most important aspects of generating a new token.

Part 2: The State Space Model (SSM)

➢ What is a State Space Model?

- **SSMs** are models used to describe these *state representations* and *make predictions of what their next state could be depending on some input*.

- Traditionally, at time *t*, SSMs:
  - ✓ map an *input sequence* $\boldsymbol{x(t)}$ — (e.g., moved left and down in the maze)
  - ✓ to **a *latent state representation*** $\boldsymbol{h(t)}$ — (e.g., distance to exit and x/y coordinates)
  - ✓ derive *a predicted output sequence* $\boldsymbol{y(t)}$ — (e.g., move left again to reach the exit sooner)

- However, instead of using discrete sequences (like moving left once) it takes as input a continuous sequence and predicts the output sequence.



- SSMs assume that dynamic systems, such as an object moving in 3D space, can be predicted from its state at time *t* through two equations.

State equation $\quad \mathbf{h'}(t) = \mathbf{A}\mathbf{h}(t) + \mathbf{B}\mathbf{x}(t)$

Output equation $\quad \mathbf{y}(t) = \mathbf{C}\mathbf{h}(t) + \mathbf{D}\mathbf{x}(t)$

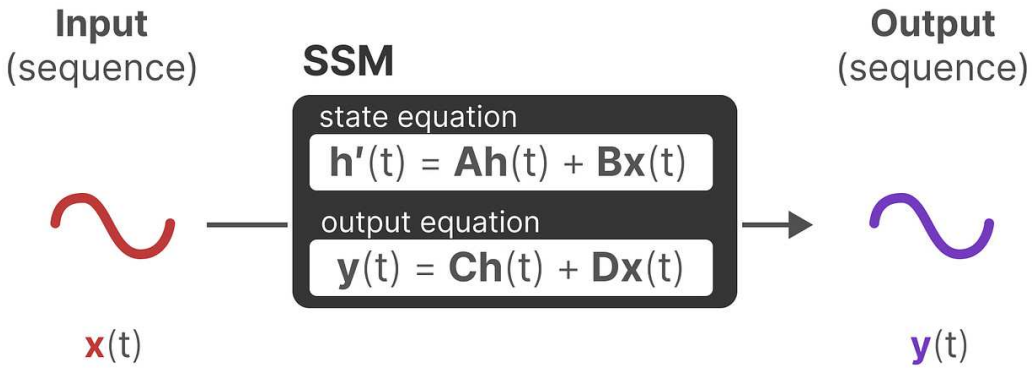
Part 2: The State Space Model (SSM)

➢ What is a State Space Model?

State equation $\mathbf{h'}(t) = \mathbf{A}\mathbf{h}(t) + \mathbf{B}\mathbf{x}(t)$

Output equation $\mathbf{y}(t) = \mathbf{C}\mathbf{h}(t) + \mathbf{D}\mathbf{x}(t)$

- By solving these equations, we assume that we can uncover the statistical principles to predict the state of a system based on observed data (*input sequence x(t)* and *previous state h(t)*).
- Its goal is to find this state representation $h(t)$ such that we can go from an input to an output sequence.



- These two equations are the core of the State Space Model.

The two equations will be referenced throughout this guide. To make them a bit more intuitive, they are color-coded so you can quickly reference them.

Part 2: The State Space Model (SSM)

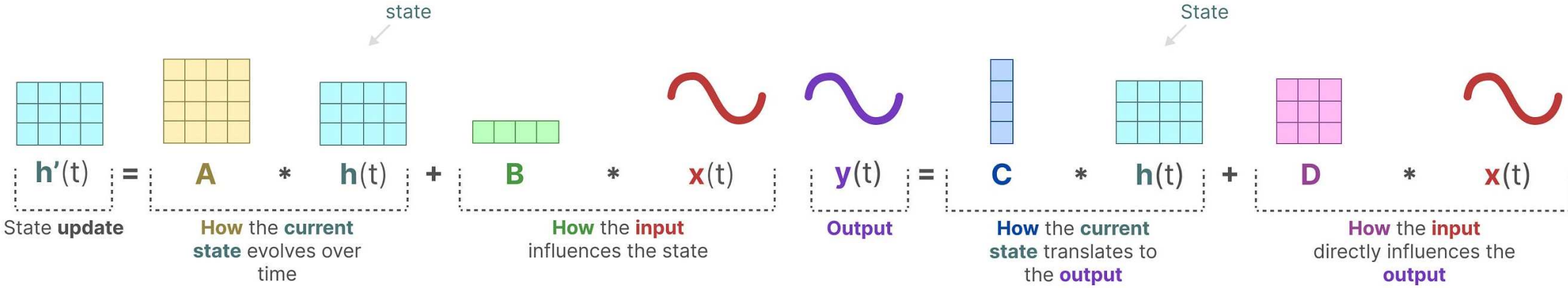
➢ What is a State Space Model?

State equation $\mathbf{h'}(t) = \mathbf{A}\mathbf{h}(t) + \mathbf{B}\mathbf{x}(t)$

- The **state equation** describes <u>how the state changes (through matrix A) based on how the input influences the state (through matrix B).</u>

Output equation $\mathbf{y}(t) = \mathbf{C}\mathbf{h}(t) + \mathbf{D}\mathbf{x}(t)$

- The **output equation** describes <u>how the state is translated to the output (through matrix C) and how the input influences the output (through matrix D).</u>



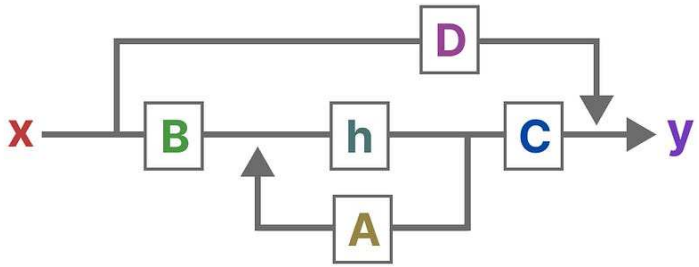
- As we saw before, $h(t)$ refers to *our latent state representation at any given time* t, and $x(t)$ refers to some input.

- NOTE: Matrices A, B, C, and D are also commonly referred to as *parameters* since they are *learnable*.

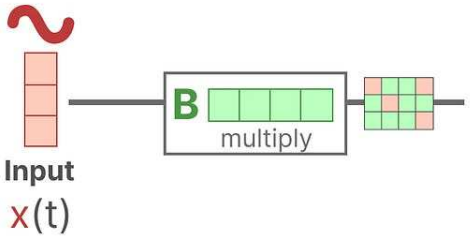
Part 2: The State Space Model (SSM)

❖ Let's go through the general technique step-by-step to understand **how these matrices influence the learning process**.

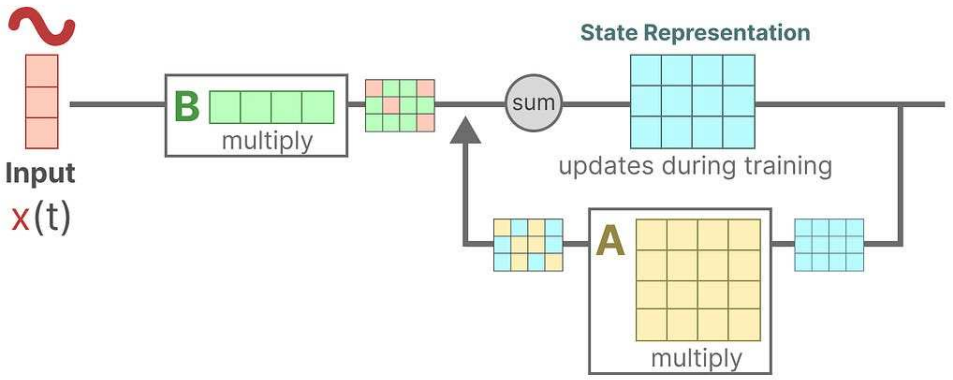
- Visualizing these two equations gives us the following architecture:



1) Assume we have some input signal $x(t)$, this signal first gets multiplied by ***matrix B*** which **describes how the inputs influence the system**.



2) The *updated state* (akin to the hidden state of a neural network) is *a latent space* that contains the core "***knowledge***" of the environment. We multiply the state with ***matrix A*** which **describes how all the internal states are connected as they represent the underlying dynamics of the system**.



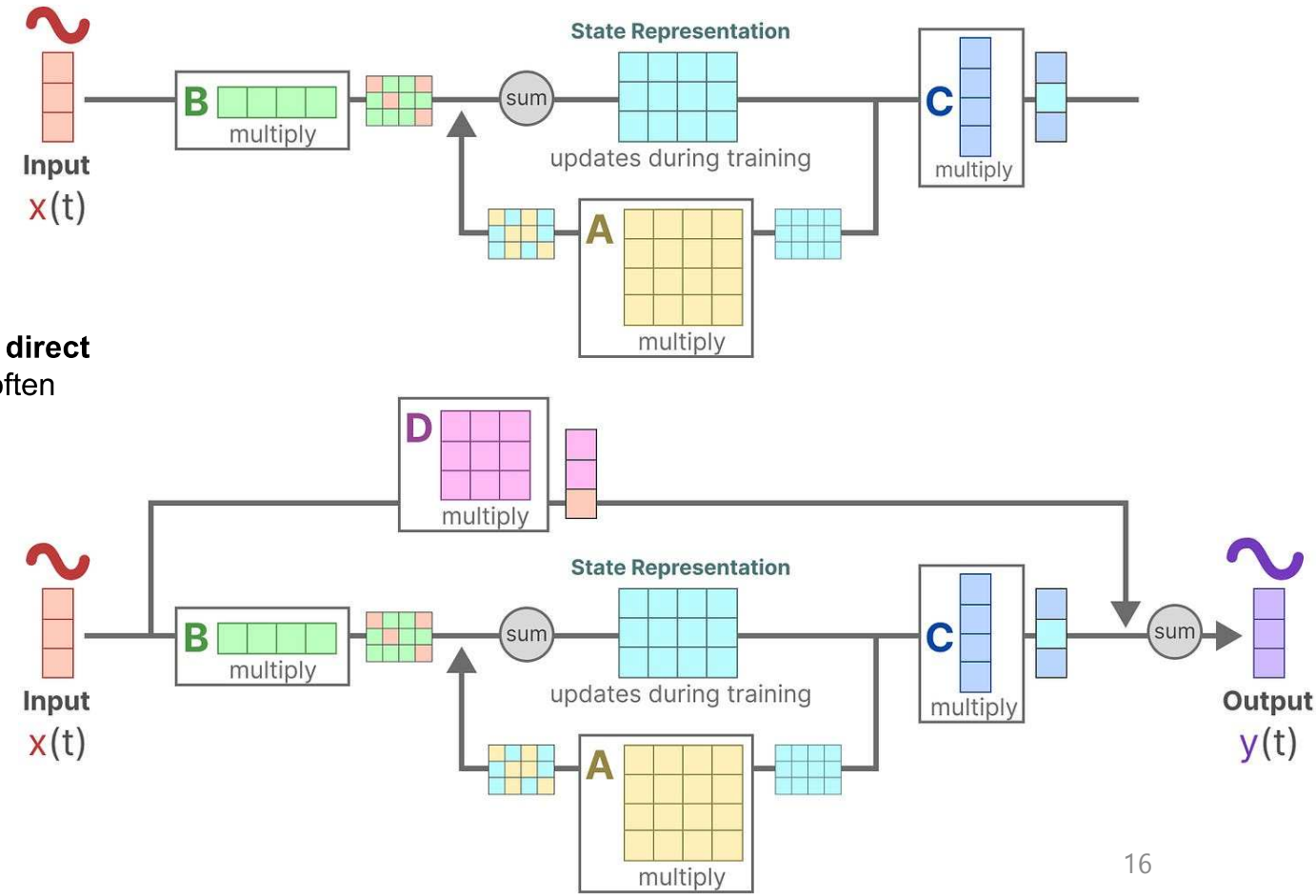
- As you might have noticed, *matrix* A is applied before creating *the state representations* and is updated after *the state representation* has been updated.

Part 2: The State Space Model (SSM)

❖ Let's go through the general technique step-by-step to understand **how these matrices influence the learning process**.

3) Then, we use ***matrix* C** to **describe how the state can be translated to an output.**

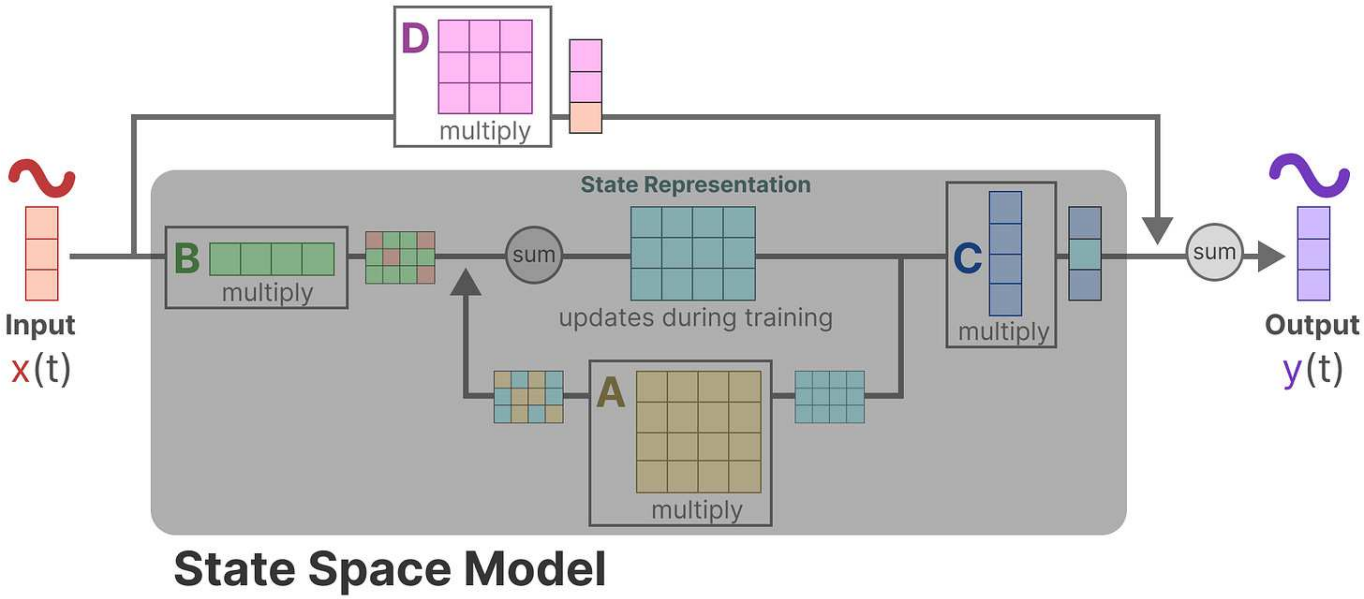
4) Finally, we can make use of ***matrix* D** to **provide a direct signal from the input to the output**. This is also often referred to as *a* ***skip-connection****.*



Part 2: The State Space Model (SSM)

❖ Let's go through the general technique step-by-step to understand **how these matrices influence the learning process**.

- Since *matrix* D is similar to a *skip-connection*, the SSM is often regarded as the following without the skip-connection.

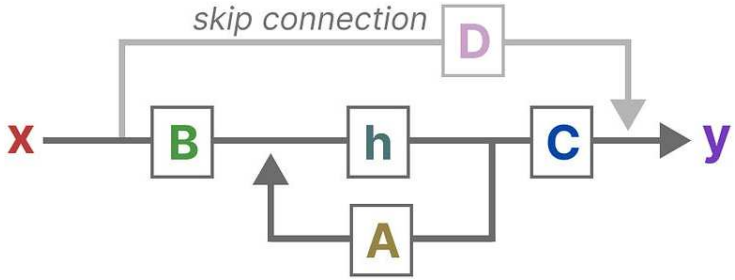


State Space Model

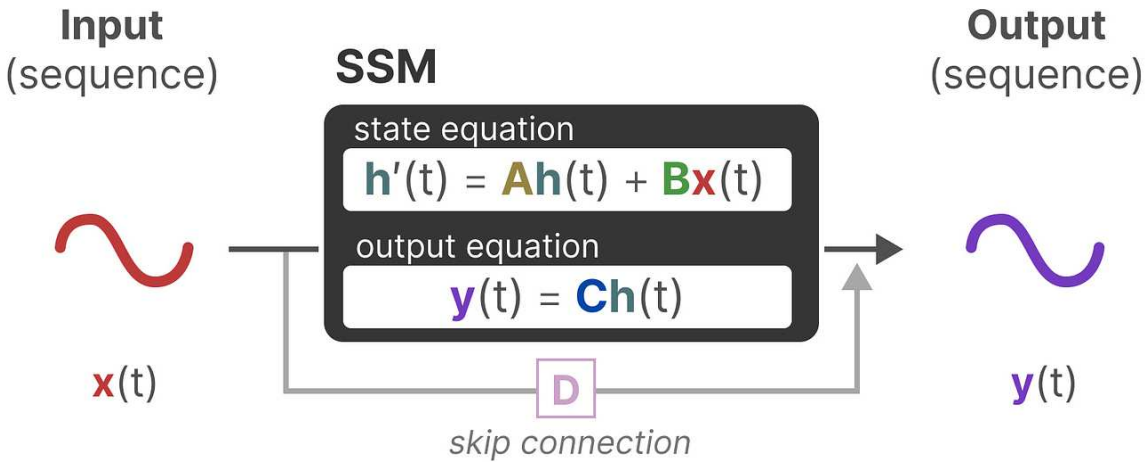
Part 2: The State Space Model (SSM)

➢ What is a State Space Model?

- Going back to our simplified perspective, we can now focus on *matrices* A, B, and C as the core of the SSM.



- We can update the original equations (and add some pretty colors) to signify the purpose of each matrix as we did before.

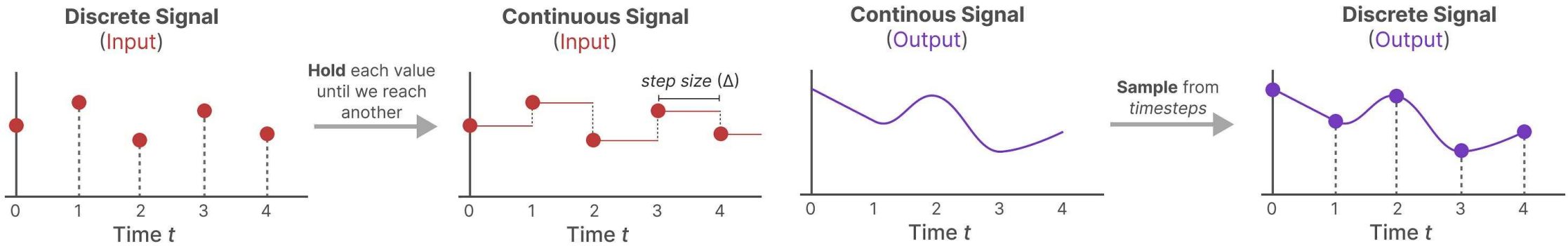


- Together, **these two equations aim to predict the state of a system from observed data**. Since the input is expected to be continuous, the main representation of the SSM is a continuous-time representation.

Part 2: The State Space Model (SSM)

➢ From a Continuous to a Discrete Signal

- Finding *the state representation* $h(t)$ is analytically challenging if you have a continuous signal. Moreover, since we generally have a discrete input (like a textual sequence), we want to discretize the model.
- To do so, we make use of the ***Zero-order hold technique***.
- It works as follows. First, every time we receive a discrete signal, we hold its value until we receive a new discrete signal. This process creates a continuous signal the SSM can use:
- How long we hold the value is represented by a *new learnable parameter*, called ***the step size*** Δ. It represents *the resolution of the input.*
- Now that we have a continuous signal for our input, we can generate a continuous output and only sample the values according to the time steps of the input.
- These sampled values are our discretized output!



Part 2: The State Space Model (SSM)

➢ From a Continuous to a Discrete Signal

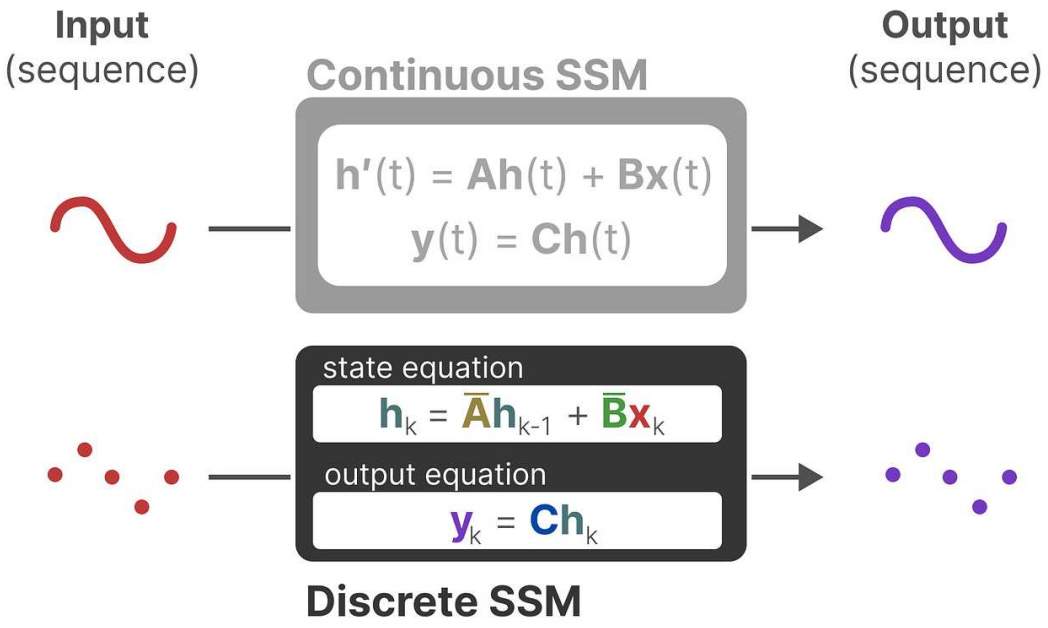
- Mathematically, we can apply *the **Zero-order hold (ZOH)*** as follows:

Discretized matrix **A** $\quad \bar{\mathbf{A}} = \exp(\Delta\mathbf{A})$

Discretized matrix **B** $\quad \bar{\mathbf{B}} = (\Delta\mathbf{A})^{-1}(\exp(\Delta\mathbf{A}) - I)\cdot \Delta\mathbf{B}$

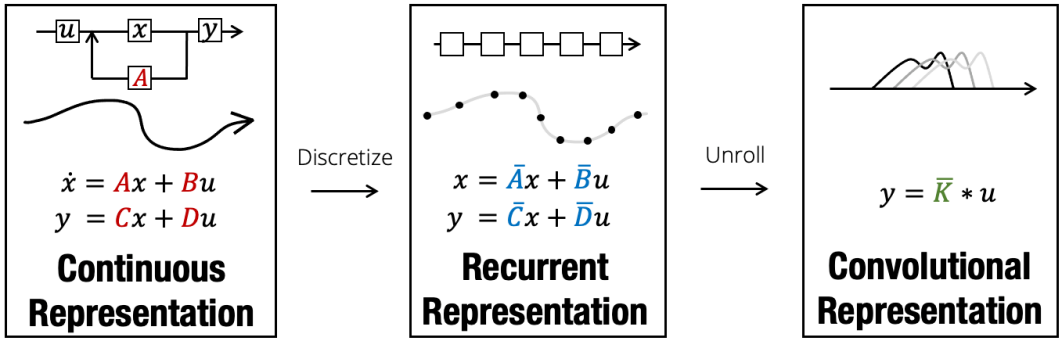
- Here, *matrices* A and B now represent discretized parameters of the model.
- We use *k* instead of *t* to represent discretized timesteps and to make it a bit more clear when we refer to a continuous versus a discrete SSM.
- **NOTE**: We are still saving the continuous form of *Matrix* A and not the discretized version during training. During training, the continuous representation is discretized.

- Together, they allow us to go from a **continuous SSM** to a **discrete SSM** represented by a formulation that instead of a function-to-function, $x(t) \rightarrow y(t)$, is now a sequence-to-sequence, $x_k \rightarrow y_k$:



Part 2: The State Space Model (SSM)

➢ From a Continuous to a Discrete Signal



$$x'(t) = \mathbf{A}x(t) + \mathbf{B}u(t) \qquad x' = \mathbf{A}x + \mathbf{B}u$$
$$y(t) = \mathbf{C}x(t) + \mathbf{D}u(t) \qquad y = \mathbf{C}x$$

$\mathbf{A} \in \mathbb{C}^{n\times n}$ is the state matrix (controlling the latent state x),

$\mathbf{B} \in \mathbb{C}^{n\times m}$ is the control matrix,

$\mathbf{C} \in \mathbb{C}^{p\times n}$ is the output matrix,

$\mathbf{D} \in \mathbb{C}^{p\times m}$ is the command matrix,

☞ Introduction to State Space Models (SSM), https://huggingface.co/blog/lbourdois/get-on-the-ssm-train

❖ Recursive view of an SSM - *Trapezoid method*

To discretize the continuous case, let's use the trapezoid method where the principle is to assimilate the region under the representative curve of a function f defined on a segment $[t_n, t_{n+1}]$ to a trapezoid and calculate its area $T : T = (t_{n+1} - t_n)\frac{f(t_n)+f(t_{n+1})}{2}$.

We then have: $x_{n+1} - x_n = \frac{1}{2}\Delta(f(t_n) + f(t_{n+1}))$ with $\Delta = t_{n+1} - t_n$.
If $x'_n = \mathbf{A}x_n + \mathbf{B}u_n$ (first line of the SSM equation), corresponds to f, so:

$$x_{n+1} = x_n + \frac{\Delta}{2}(\mathbf{A}x_n + \mathbf{B}u_n + \mathbf{A}x_{n+1} + \mathbf{B}u_{n+1})$$
$$\iff x_{n+1} - \frac{\Delta}{2}\mathbf{A}x_{n+1} = x_n + \frac{\Delta}{2}\mathbf{A}x_n + \frac{\Delta}{2}\mathbf{B}(u_{n+1} + u_n)$$
$$(*) \iff (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})x_{n+1} = (\mathbf{I} + \frac{\Delta}{2}\mathbf{A})x_n + \Delta\mathbf{B}u_{n+1}$$
$$\iff x_{n+1} = (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})^{-1}(\mathbf{I} + \frac{\Delta}{2}\mathbf{A})x_n + (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})^{-1}\Delta\mathbf{I}$$

(*) $u_{n+1} \overset{\Delta}{\simeq} u_n$ (the control vector is assumed to be constant over a small Δ).

Part 2: The State Space Model (SSM)

➢ From a Continuous to a Discrete Signal

❖ Trapezoid method

We've just obtained our discretized SSM!

To make this completely explicit, let's pose:

$$\bar{\mathbf{A}} = (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})^{-1}(\mathbf{I} + \frac{\Delta}{2}\mathbf{A})$$
$$\bar{\mathbf{B}} = (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})^{-1}\Delta\mathbf{B}$$
$$\bar{\mathbf{C}} = \mathbf{C}$$

We then have

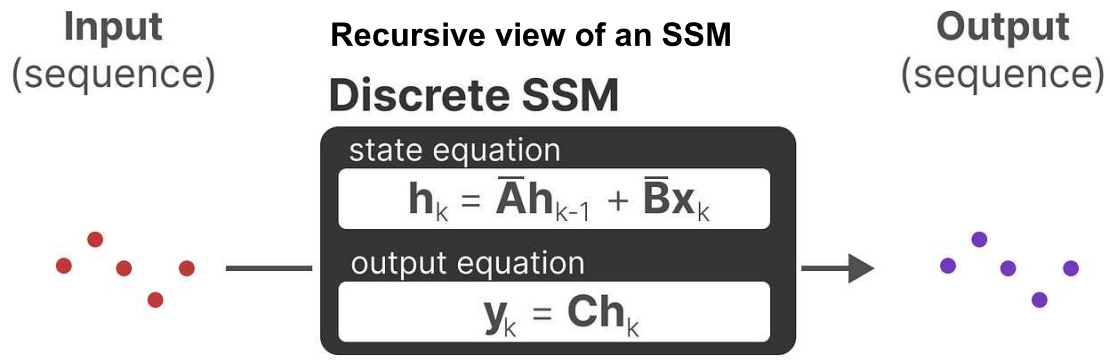
$$x_k = \bar{\mathbf{A}}x_{k-1} + \bar{\mathbf{B}}u_k$$
$$y_k = \bar{\mathbf{C}}x_k$$

The notation of matrices with a bar was introduced in S4 to designate matrices in the discrete case and has since become a convention in the field of SSM applied to deep learning.

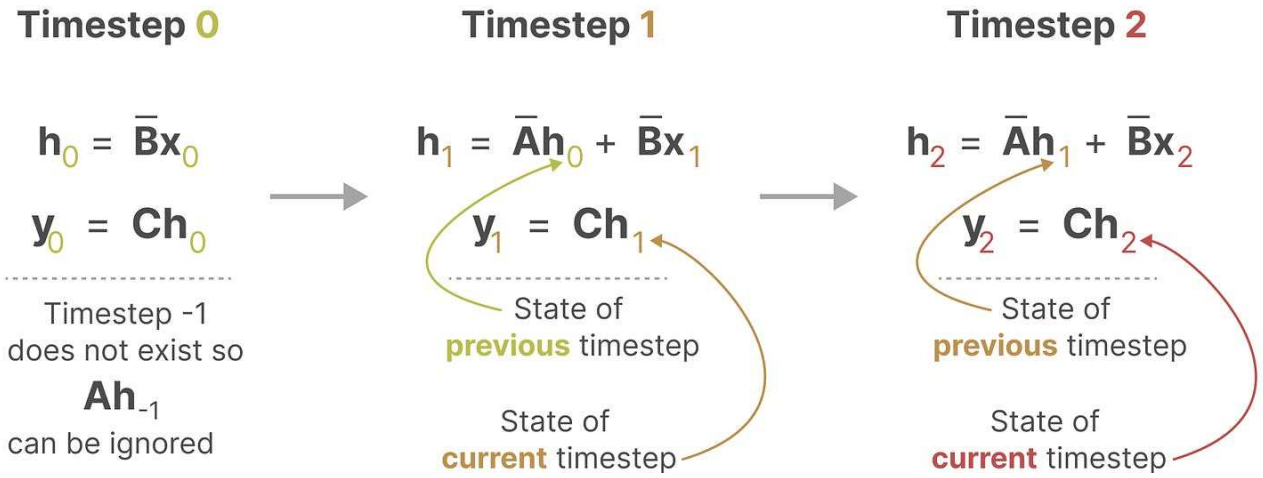
Part 2: The State Space Model (SSM)

➢ The Recurrent Representation

- Our discretized SSM allows us to formulate the problem in specific timesteps instead of continuous signals. A recurrent approach, as we saw before with RNNs is quite useful here.
- If we consider discrete timesteps instead of a continuous signal, we can reformulate the problem with timesteps:



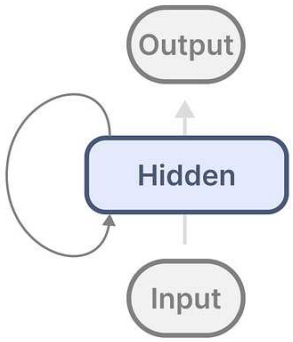
- At each timestep, we calculate how the current input ($\bar{B}x_k$) influences the previous state ($\bar{A}h_{k-1}$) and then calculate the predicted output ($\bar{C}h_k$).



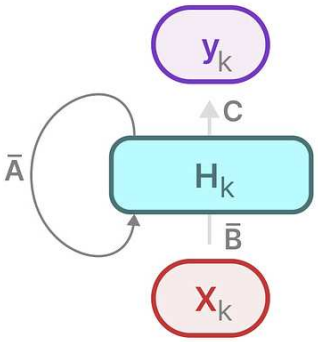
Part 2: The State Space Model (SSM)

➢ The Recurrent Representation

- This representation might already seem a bit familiar! We can approach it the same way we did with the RNN as we saw before.

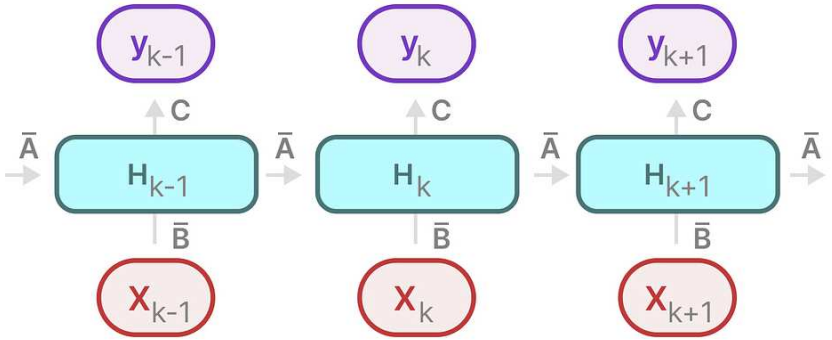


RNN



SSM
(Recurrent)

- Which we can unfold (or unroll) as such:

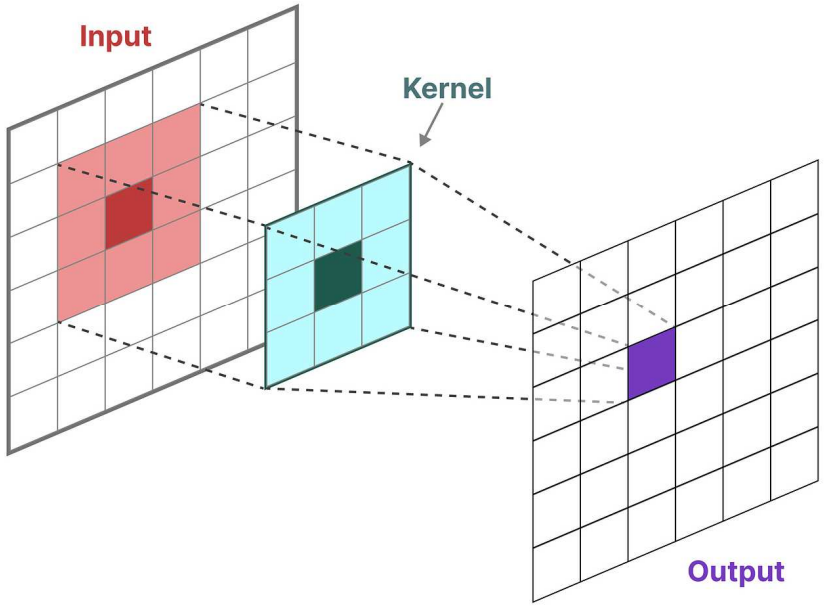


SSM
(Recurrent + Unfolded)

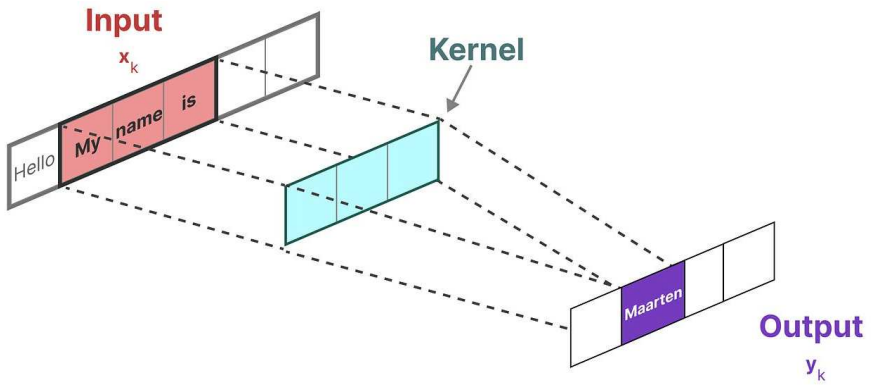
Part 2: The State Space Model (SSM)

➢ The Convolution Representation

- Another representation that we can use for SSMs is that of *convolutions*. Remember from classic image recognition tasks where we applied filters (kernels) to derive aggregate features:



- Since we are dealing with text and not images, we need a 1-dimensional perspective instead:



- **The Convolution Representation**

$$\mathbf{h}_0 = \bar{\mathbf{B}}\mathbf{x}_0 \qquad \mathbf{h}_1 = \bar{\mathbf{A}}\mathbf{h}_0 + \bar{\mathbf{B}}\mathbf{x}_1 \qquad \mathbf{h}_1 = \bar{\mathbf{A}}\mathbf{h}_1 + \bar{\mathbf{B}}\mathbf{x}_2$$

$$\mathbf{y}_0 = \mathbf{C}\mathbf{h}_0 \qquad \mathbf{y}_1 = \mathbf{C}\mathbf{h}_1 \qquad \mathbf{y}_2 = \mathbf{C}\mathbf{h}_2$$

$$= \mathbf{C}\bar{\mathbf{B}}\mathbf{x}_0 \qquad = \mathbf{C}(\bar{\mathbf{A}}\mathbf{h}_0 + \bar{\mathbf{B}}\mathbf{x}_1) \qquad = \mathbf{C}(\bar{\mathbf{A}}\mathbf{h}_1 + \bar{\mathbf{B}}\mathbf{x}_2)$$

$$= \mathbf{C}\bar{\mathbf{A}}\bar{\mathbf{B}}\mathbf{x}_0 + \mathbf{C}\bar{\mathbf{B}}\mathbf{x}_1 \qquad = \mathbf{C}\bar{\mathbf{A}}^2\bar{\mathbf{B}}\mathbf{x}_0 + \mathbf{C}\bar{\mathbf{A}}\bar{\mathbf{B}}\mathbf{x}_1 + \mathbf{C}\bar{\mathbf{B}}\mathbf{x}_2$$

$$\mathbf{y}_0 = \mathbf{C}\bar{\mathbf{B}}\mathbf{x}_0$$

$$\mathbf{y}_1 = \mathbf{C}\bar{\mathbf{A}}\bar{\mathbf{B}}\mathbf{x}_0 + \mathbf{C}\bar{\mathbf{B}}\mathbf{x}_1$$

$$\mathbf{y}_2 = \mathbf{C}\bar{\mathbf{A}}^2\bar{\mathbf{B}}\mathbf{x}_0 + \mathbf{C}\bar{\mathbf{A}}\bar{\mathbf{B}}\mathbf{x}_1 + \mathbf{C}\bar{\mathbf{B}}\mathbf{x}_2 \qquad \mathbf{k} = (\mathbf{C}\bar{\mathbf{B}}, \mathbf{C}\bar{\mathbf{A}}\bar{\mathbf{B}}, \mathbf{C}\bar{\mathbf{A}}^2\bar{\mathbf{B}})$$

$$\mathbf{y} = \mathbf{x} * \mathbf{k}$$

Part 2: The State Space Model (SSM)

➢ The Convolution Representation

☞ https://huggingface.co/blog/lbourdois/get-on-the-ssm-train

This recurrence can be written as a convolution. To do this, simply iterate the equations of the system

$$x_k = \bar{\mathbf{A}} x_{k-1} + \bar{\mathbf{B}} u_k$$
$$y_k = \bar{\mathbf{C}} x_k$$

Let's start with the first line of the system:

Step 0: $x_0 = \bar{\mathbf{B}} u_0$

Step 1: $x_1 = \bar{\mathbf{A}} x_0 + \bar{\mathbf{B}} u_1 = \bar{\mathbf{A}}\bar{\mathbf{B}} u_0 + \bar{\mathbf{B}} u_1$

Step 2: $x_2 = \bar{\mathbf{A}} x_1 + \bar{\mathbf{B}} u_2 = \bar{\mathbf{A}}(\bar{\mathbf{A}}\bar{\mathbf{B}} u_0 + \bar{\mathbf{B}} u_1) + \bar{\mathbf{B}} u_2 = \bar{\mathbf{A}}^2\bar{\mathbf{B}} u_0 + \bar{\mathbf{A}}\bar{\mathbf{B}} u_1 + \bar{\mathbf{B}} u_2$

We have x_k which can be written as a function f parametrized by $(u_0, u_1, ...u_k)$.

Let's move on to the second line of the system, where we can now inject the x_k values calculated just now:

Step 0: $y_0 = \bar{\mathbf{C}} x_0 = \bar{\mathbf{C}}\bar{\mathbf{B}} u_0$

Step 1: $y_1 = \bar{\mathbf{C}} x_1 = \bar{\mathbf{C}}(\bar{\mathbf{A}}\bar{\mathbf{B}} u_0 + \bar{\mathbf{B}} u_1) = \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}} u_0 + \bar{\mathbf{C}}\bar{\mathbf{B}} u_1$

Step 2: $y_2 = \bar{\mathbf{C}} x_2 = \bar{\mathbf{C}}(\bar{\mathbf{A}}^2\bar{\mathbf{B}} u_0 + \bar{\mathbf{A}}\bar{\mathbf{B}} u_1 + \bar{\mathbf{B}} u_2) = \bar{\mathbf{C}}\bar{\mathbf{A}}^2\bar{\mathbf{B}} u_0 + \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}} u_1 + \bar{\mathbf{C}}\bar{\mathbf{B}} u_2$

We can observe the convolution kernel $\bar{\mathbf{K}}_k = (\bar{\mathbf{C}}\bar{\mathbf{B}}, \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}}, ..., \bar{\mathbf{C}}\bar{\mathbf{A}}^k\bar{\mathbf{B}})$ applicable to u_k, hence $K * u$.

As with matrices, we apply a bar to the $\bar{\mathbf{K}}$ to specify that it is the convolution kernel obtained after discretization. It is generally referred to as the **SSM convolution kernel** in the literature, and its size is equivalent to the entire input sequence. This convolution kernel is calculated by Fast Fourier Transform (FFT) and will be explained in future articles.

Part 2: The State Space Model (SSM)

➢ The Convolution Representation

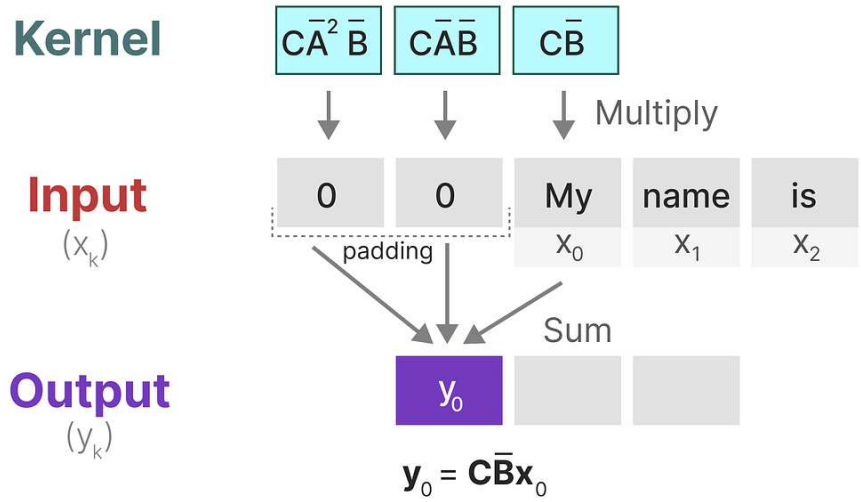
- The kernel that we use to represent this “filter” is derived from the SSM formulation:

$$\text{kernel} \longrightarrow \overline{\mathbf{K}} = (\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{A}\mathbf{B}}, \ldots, \mathbf{C}\overline{\mathbf{A}}^{K}\overline{\mathbf{B}}, \ldots)$$

$$\mathbf{y} = \mathbf{x} * \overline{\mathbf{K}}$$

output input kernel

- Let’s explore how this kernel works in practice. Like convolution, we can use our SSM kernel to go over each set of tokens and calculate the output:

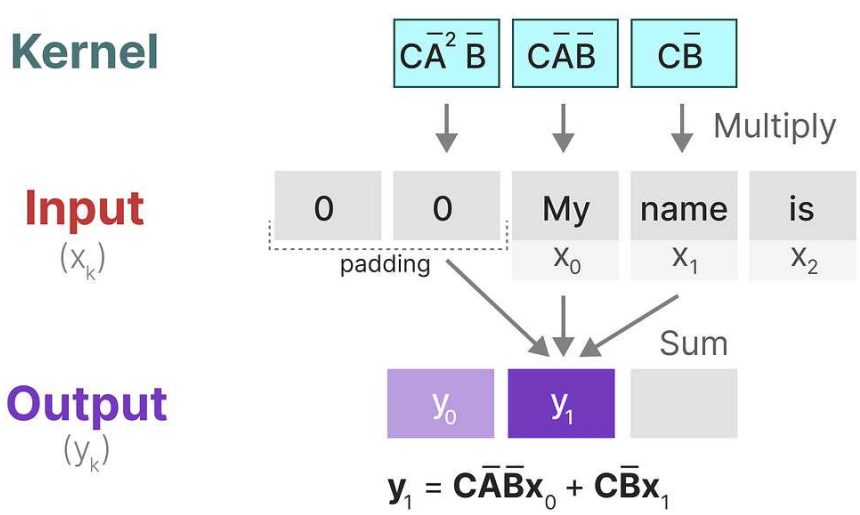


- This also illustrates the effect padding might have on the output. I changed the order of padding to improve the visualization but we often apply it at the end of a sentence.

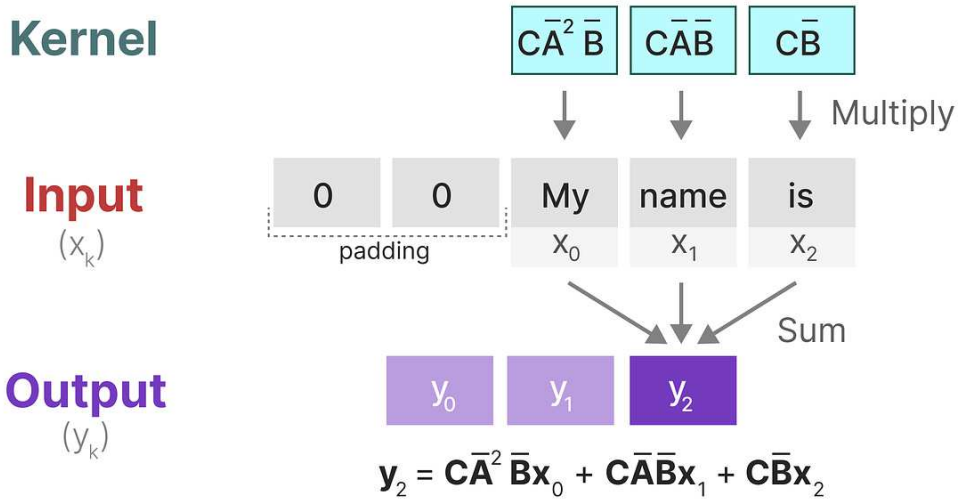
Part 2: The State Space Model (SSM)

➢ The Convolution Representation

- In the next step, the kernel is moved once over to perform the next step in the calculation:



- In the final step, we can see the full effect of the kernel:

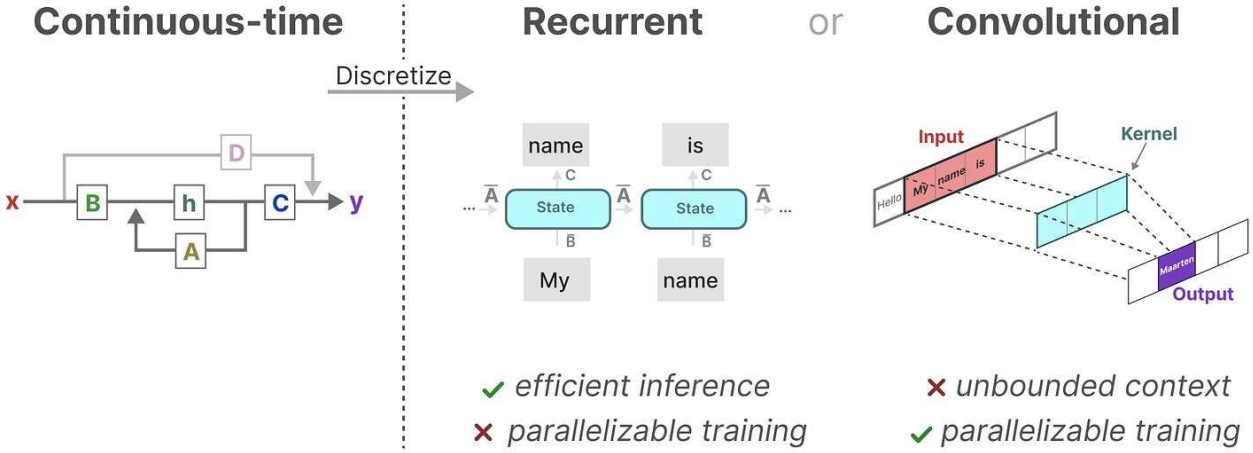


❖ A major benefit of representing the SSM as a convolution is that **it can be trained in parallel like Convolutional Neural Networks (CNNs)**. However, **due to the fixed kernel size, their inference is not as fast and unbounded as RNNs.**

Part 2: The State Space Model (SSM)

➢ The Three Representations

- These three representations, ***continuous***, ***recurrent***, and ***convolutional*** all have different sets of advantages and disadvantages:

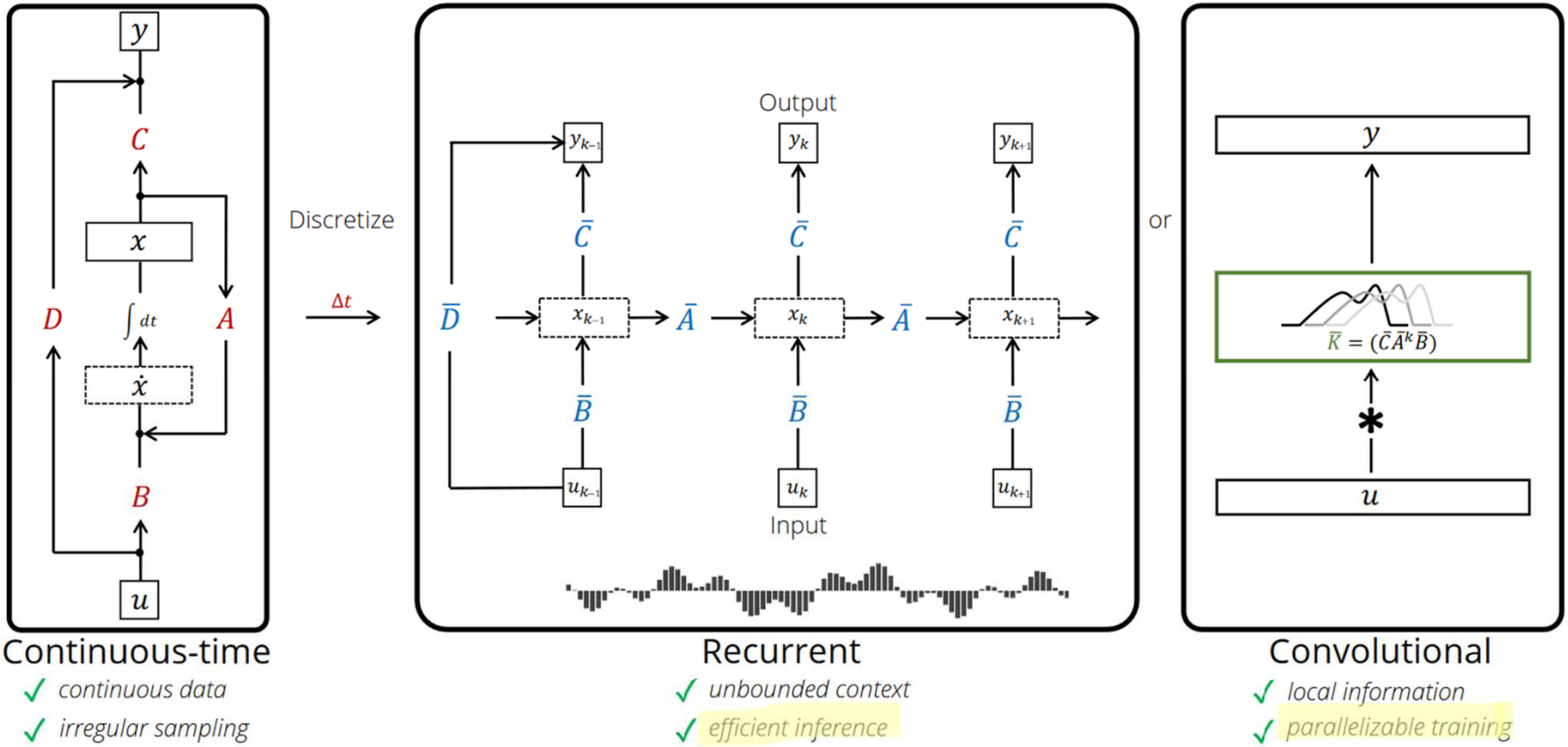


- Interestingly, we now have efficient inference with the recurrent SSM and parallelizable training with the convolutional SSM.

Part 2: The State Space Model (SSM)

➢ **The Three Representations**

☞ https://huggingface.co/blog/lbourdois/get-on-the-ssm-train



✓ Mathematically feasible analysis

✗ Extremely slow for training & inference.

✗ Slow learning (lack of parallelism).

✗ Gradient disappearance or explosion when training too-long sequences.

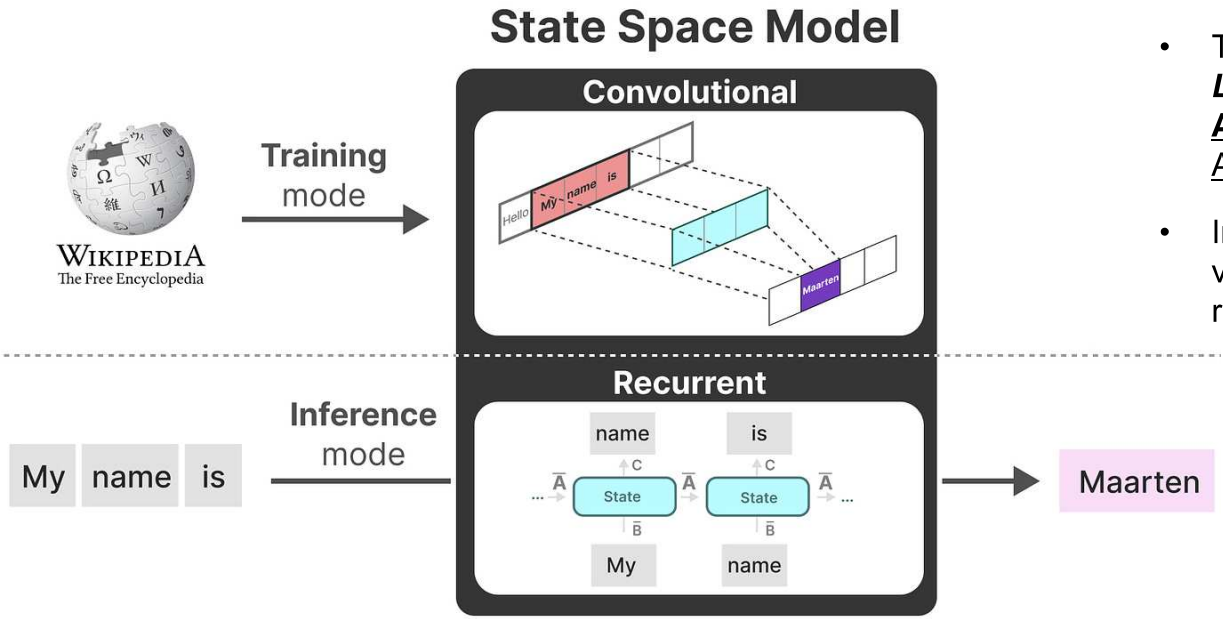
✗ Slowness in online or autoregressive contexts (must recalculate entire input for each new data point).

✗ Fixed context size.

Part 2: The State Space Model (SSM)

➢ The Three Representations

- With these representations, there is a neat trick that we can use, namely choose a representation depending on the task. During **training**, we use **the convolutional representation** which can be **parallelized** and during **inference**, we use the **efficient recurrent representation**:



- This model is referred to as *the Linear State-Space Layer* (LSSL).

➢ Gu, Albert, et al. "Combining recurrent, convolutional, and continuous-time models with linear state space layers." Advances in neural information processing systems 34 (2021): 572-585.

- These representations share an important property, namely that of ***Linear Time Invariance*** (***LTI***). LTI states that **the SSMs parameters, A, B, and C, are fixed for all timesteps**. This means that matrices A, B, and C are the same for every token the SSM generates.
- In other words, regardless of what sequence you give the SSM, the values of A, B, and C remain the same. We have a static representation that is not content-aware.

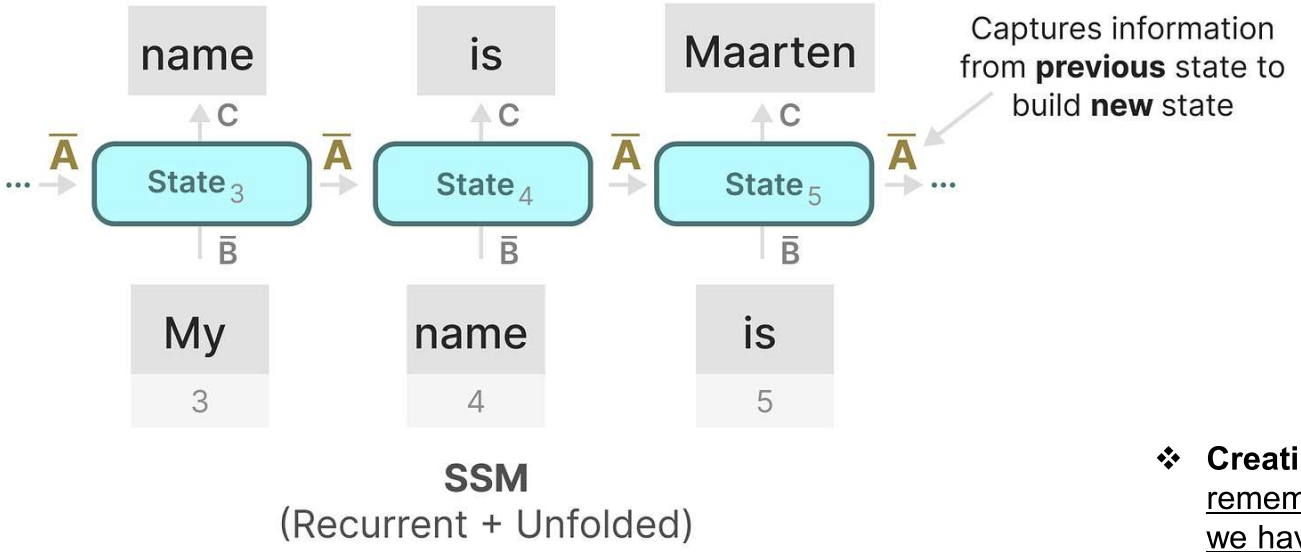
➢ Before we explore how Mamba addresses this issue, let's explore the final piece of the puzzle, matrix A.

[Memo] LTI system : 선형성(Linearity)과 시불변성(Time invariant)을 모두 가진 시스템이다. 선형 시스템은 입력과 출력이 선형성을 만족시키는 시스템을 말한다. 시불변 시스템은 입력에서 시간의 지연 또는 선행이 있을 때, 출력에도 같은 시간지연이 있는 시스템을 의미한다.

Part 2: The State Space Model (SSM)

➢ The Importance of Matrix A

- Arguably one of the most important aspects of the SSM formulation is ***matrix* A**. As we saw before with the recurrent representation, it **captures information about the previous state to build the new state**.



- In essence, *matrix* A produces the hidden state:

Produces hidden state

$$h_k = \bar{A}h_{k-1} + \bar{B}x_k$$

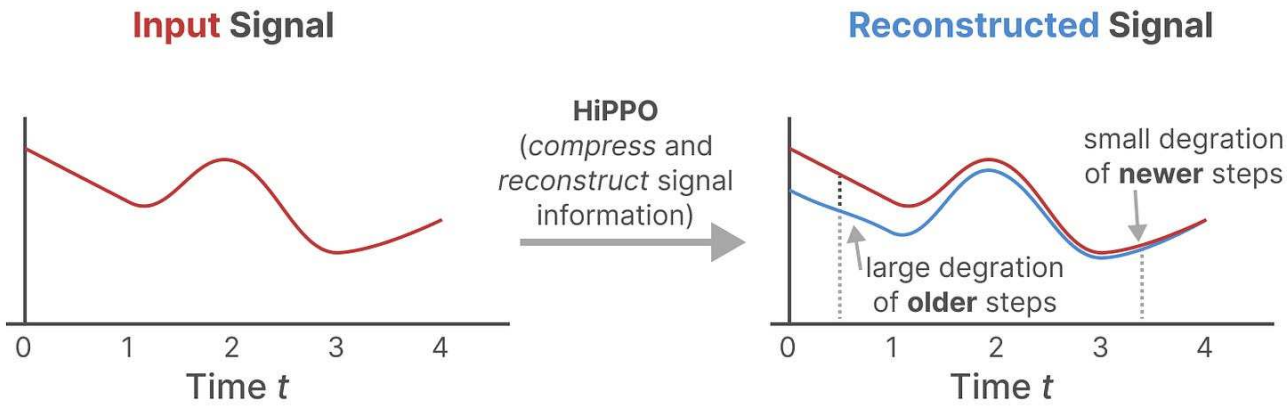
$$y_k = Ch_k$$

❖ **Creating *matrix* A** can therefore be the difference between remembering only a few previous tokens and capturing every token we have seen thus far. Especially in the context of the Recurrent representation since it only looks back at the previous state.

Part 2: The State Space Model (SSM)

➢ The Importance of Matrix A

- So how can we create *matrix* A in a way that retains a large memory (context size)?
- We use **Hungry Hungry Hippo**! Or **HiPPO** for **H**igh-order **P**olynomial **P**rojection **O**perators. HiPPO attempts to **compress all input signals** it has seen thus far into a vector of coefficients.
  - ➢ Gu, Albert, et al. "Hippo: Recurrent memory with optimal polynomial projections." Advances in neural information processing systems 33 (2020): 1474-1487.



- It uses ***matrix* A** to **build a state representation** that **captures recent tokens well** and **decays older tokens**. Its formula can be represented as follows:

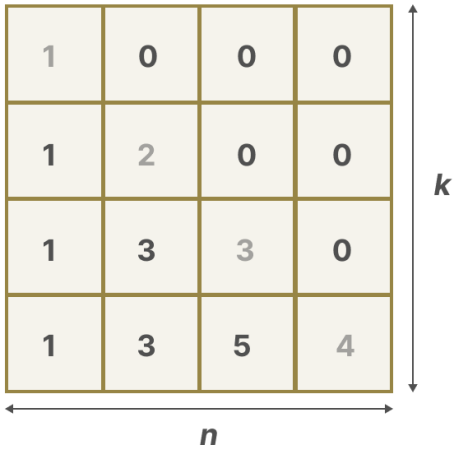
$$\text{HiPPO Matrix} \quad A_{nk} = \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{everything below the diagonal} \\ n+1 & \text{the diagonal} \\ 0 & \text{everything above the diagonal} \end{cases}$$

Part 2: The State Space Model (SSM)

➢ The Importance of Matrix A

- Assuming we have a square matrix A, this gives us:

HiPPO Matrix



- Building *matrix* A using HiPPO was shown to be much better than initializing it as a random matrix. As a result, it more accurately reconstructs newer signals (recent tokens) compared to *older* signals (initial tokens).

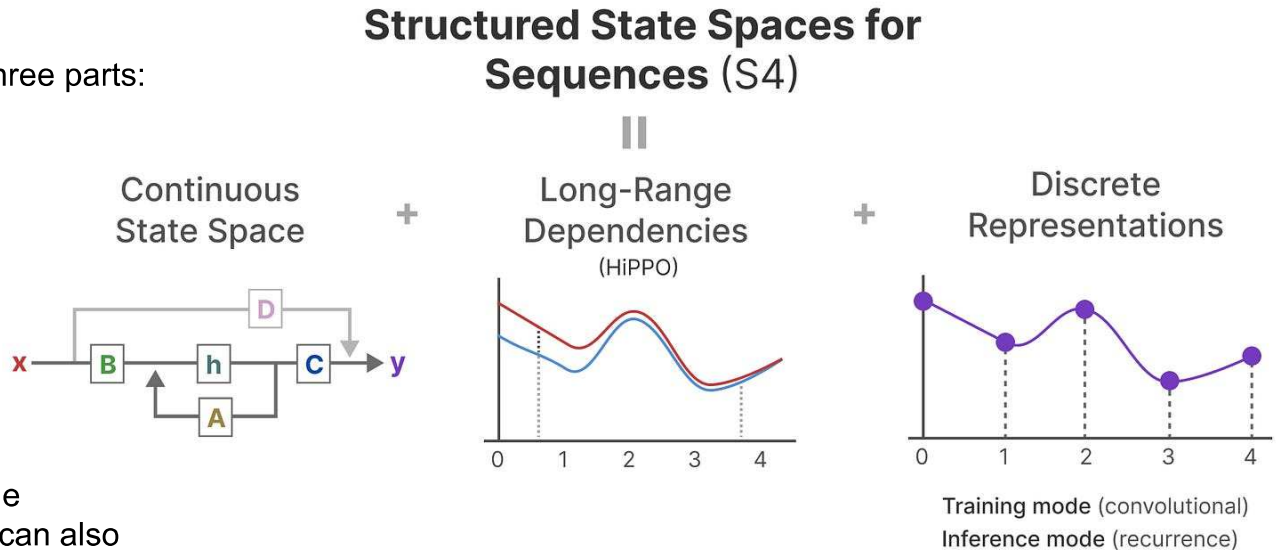
- The idea behind **the HiPPO Matrix** is that it **produces a hidden state that memorizes its history**.
- Mathematically, it does so by tracking the coefficients of a *Legendre polynomial* which allows it to approximate all of the previous history.
  - Voelker, Aaron, Ivana Kajić, and Chris Eliasmith. "Legendre memory units: Continuous-time representation in recurrent neural networks." Advances in neural information processing systems 32 (2019) https://proceedings.neurips.cc/paper/2019/hash/952285b9b7e7a1be5aa7849f32ffff05-Abstract.html

- **HiPPO** was then applied to **the recurrent and convolution representations** that we saw before to handle long-range dependencies. The result was **Structured State Space for Sequences (S4)**, a class of SSMs that can efficiently **handle long sequences**.
  - Gu, Albert, Karan Goel, and Christopher Ré. "Efficiently modeling long sequences with structured state spaces." arXiv preprint arXiv:2111.00396 (2021).

Part 2: The State Space Model (SSM)

➢ The Importance of Matrix A

- Structured State Space for Sequences (S4) consists of three parts:
  - ◆ State Space Models
  - ◆ HiPPO for handling long-range dependencies
  - ◆ Discretization for creating recurrent and convolution representations



- This class of SSMs has several benefits depending on the representation you choose (recurrent vs. convolution). It can also handle long sequences of text and store memory efficiently by building upon the HiPPO matrix.

[☞Grok] Hippo Matrix는 SSM에서 상태 전이 행렬 A를 초기화하는 데 사용되는 중요한 도구로, 특히 장거리 의존성을 포착하는 데 효과적임. 이는 S4와 같은 모델에서 성능을 크게 향상시키며, 다양한 변형과 대안이 연구되고 있음

Part 2: The State Space Model (SSM)

❖ Long-Range Dependencies with HiPPO

☞ Annotated S4 https://srush.github.io/annotated-s4/
☞ https://kyujinpy.tistory.com/146

➢ High-order Polynomial Projection Operators

2.1 HiPPO Problem Setup

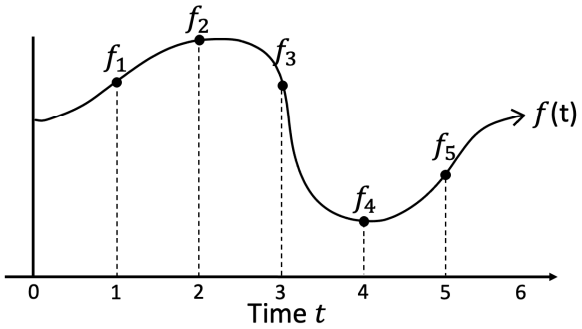
Given an input function $f(t) \in \mathbb{R}$ on $t \geq 0$, many problems require operating on the cumulative *history* $f_{\leq t} := f(x)\,|_{x \leq t}$ at every time $t \geq 0$, in order to understand the inputs seen so far and make future predictions. Since the space of functions is intractably large, the history cannot be perfectly memorized and must be compressed; we propose the general approach of projecting it onto a subspace of bounded dimension. Thus, our goal is to maintain (online) this compressed representation of the history. In order to specify this problem fully, we require two ingredients: a way to quantify the approximation, and a suitable subspace.

- Time-series function f(t)를 정의
- 이때, f(t)에 매핑되는 space는 일단 기본적으로 다루기 힘들 정도로 클 것임
- 그리고, 모든 time (history)에 대해서 기억하지 못함
- HiPPO 논문에서는, 적절한 subspace에 f(t)를 approximation하는 문제를 해결하고자 함

Part 2: The State Space Model (SSM)

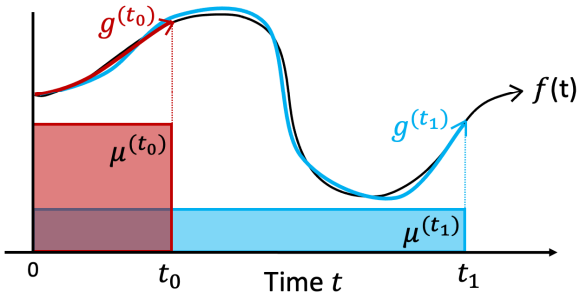
➢ **General HiPPO Framework**

☞ Annotated S4 https://srush.github.io/annotated-s4/
☞ https://kyujinpy.tistory.com/146



(1)

proj_t



(2)

coef_t

$c(t_0) = \begin{bmatrix} 0.1 \\ -1.1 \\ 3.7 \\ 2.5 \end{bmatrix}$ $c(t_1) = \begin{bmatrix} 1.5 \\ 2.9 \\ -0.3 \\ 2.0 \end{bmatrix}$

Continuous-time HiPPO ODE

$$\frac{d}{dt}c(t) = A(t)c(t) + B(t)f(t)$$

(3)

discretize

Discrete-time HiPPO Recurrence

$$c_{k+1} = A_k c_k + B_k f_k$$

(4)

(1) 함수 f(t) 정의

(2) t0, t1 , ... T에 따라 optimal하게 projection되는 polynomial function g(t)가 존재 → g(t)는 measures u(t)에 따라 결정

(2-3) g(t)를 R^N space vector를 가지는 coef.로 mapping. → (2) + (3)을 합쳐서 HiPPO 라고 부름

(3) 시간 변화율에 따른 continous-time HiPPO ODE(ordinary dynamic equation) 정의 → HiPPO ODE에서 f는 t 시점까지의 f 함수

(4) 하지만, 우리가 다루는 공간은 이산적이므로, 이를 회귀(recurrence)문제로 다시 정의 → [k+1번째의 Coef.] = [고정된 A, B, f & k번째 Coef.]의 matrix 곱셈 연산으로 이루어짐.

- 위의 HiPPO framework는 uniform measures를 활용함.

➢ 시간 함수 f(t)를 projection시키고, coeff로 매핑해서 c(t)를 찾는게 HiPPO의 framwork이라고 이해

Part 2: The State Space Model (SSM)

➢ High Order Projection: Measure Families and HiPPO ODEs

- Projection matrix를 결정할 measures에 대한 기본적인 2가지 transition

1. translated Legendre (LegT)

$$\textbf{LegT}: \mu^{(t)}(x) = \frac{1}{\theta}\mathbb{I}_{[t-\theta,t]}(x)$$

$$\textbf{LegT}: A_{nk} = \frac{1}{\theta}\begin{cases}(-1)^{n-k}(2n+1) & if\ n \geq k \\ 2n+1 & if\ n \leq k\end{cases}, \quad B_n = \frac{1}{\theta}(2n+1)(-1)^n \quad (1)$$

2. translated Laguerre (LagT)

$$\textbf{LagT}: \mu^{(t)}(x) = e^{-(t-x)}\mathbb{I}_{(-\infty,t]}(x) = \begin{cases}e^{x-t} & \text{if } x \leq t \\ 0 & \text{if } x > t\end{cases}$$

$$\textbf{LagT}: A_{nk} = \begin{cases}1 & if\ n \geq k \\ 0 & if\ n < k\end{cases}, \quad B_n = 1 \quad (2)$$

➢ HiPPO-LegS: Scaled Measures for Timescale Robustness

- 위의 measures에서 더 memory적으로 효율적으로 tight하게 approximation이 가능한 **scaled Legendre measure (LegS)** 제안함 (사실 이게 main measures!)

$$\frac{d}{dt}c(t) = -\frac{1}{t}Ac(t) + \frac{1}{t}Bf(t) \quad (3)$$

$$c_{k+1} = \left(1 - \frac{A}{k}\right)c_k + \frac{1}{k}Bf_k \quad (4)$$

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

Part 3: Mamba - A Selective SSM

- We have covered all the fundamentals necessary to understand what makes Mamba special.
- State Space Models can be used to model textual sequences but still have a set of disadvantages we want to prevent.

- In this section, we will go through Mamba's two main contributions:
  - ✓ **A selective scan algorithm** ; allows the model to filter (ir)relevant information
  - ✓ **A hardware-aware algorithm** ; allows for efficient storage of (intermediate) results through ***parallel scan***, ***kernel fusion***, and ***recomputation***.

- Together they create the **selective SSM** or **S6** models which can be used, like self-attention, to create ***Mamba blocks***.

- Before exploring the two main contributions, let's first explore why they are necessary.

Part 3: Mamba - A Selective SSM

➢ What Problem does it attempt to Solve?

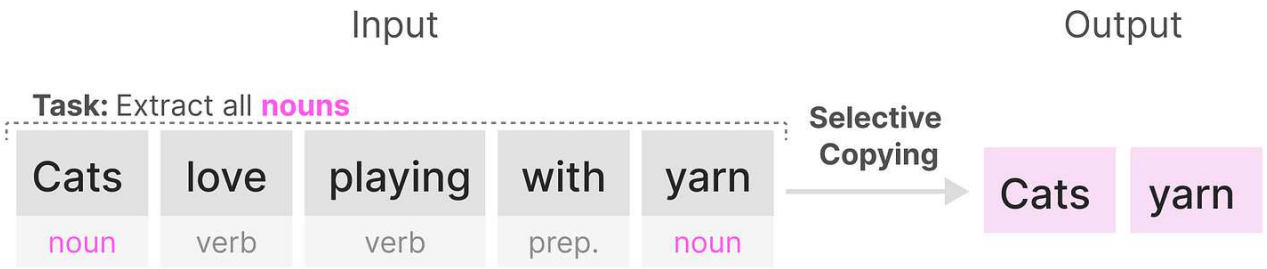
- **State Space Models**, and even the **S4** (**Structured State Space Model**), perform ***poorly*** on certain tasks that are vital in language modeling and generation, namely *the ability to focus on or ignore particular inputs.*

- We can illustrate this with two synthetic tasks, namely **selective copying** and **induction heads**.

1) In the **selective copying** task, the goal of the SSM is to copy parts of the input and output them in order:

- However, a (recurrent/convolutional) SSM performs *poorly* in this task since it is ***Linear Time Invariant***. As we saw before, the matrices A, B, and C are the same for every token the SSM generates.

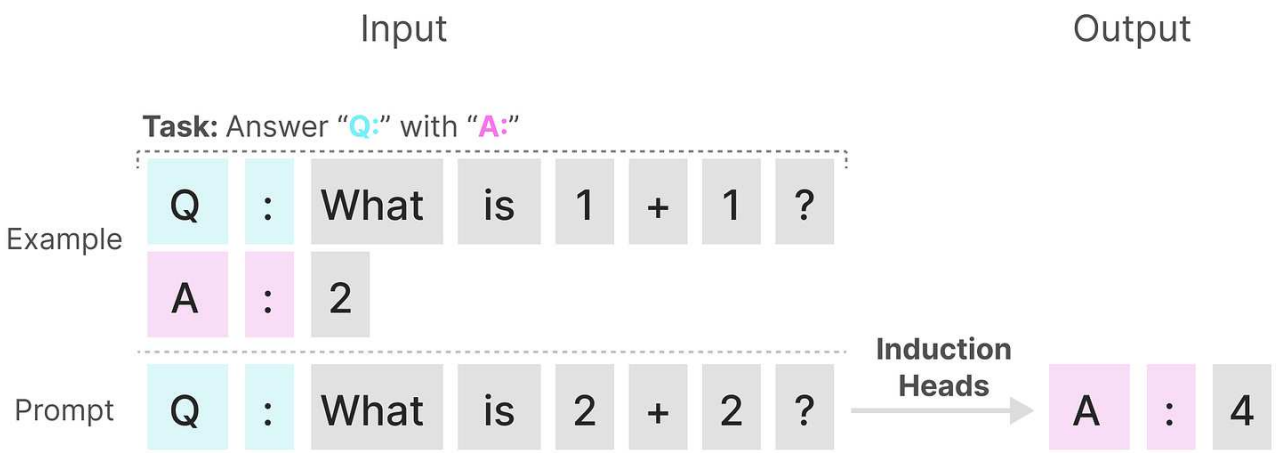
- As a result, an SSM **cannot perform *content-aware reasoning*** since it treats each token equally as a result of the fixed A, B, and C matrices. This is a problem as *we want the SSM to reason about the input (prompt)*.



Part 3: Mamba - A Selective SSM

➢ **What Problem does it attempt to Solve?**

2) The second task an SSM performs ***poorly*** on is **induction heads** where the goal is to **reproduce patterns found in the input**:

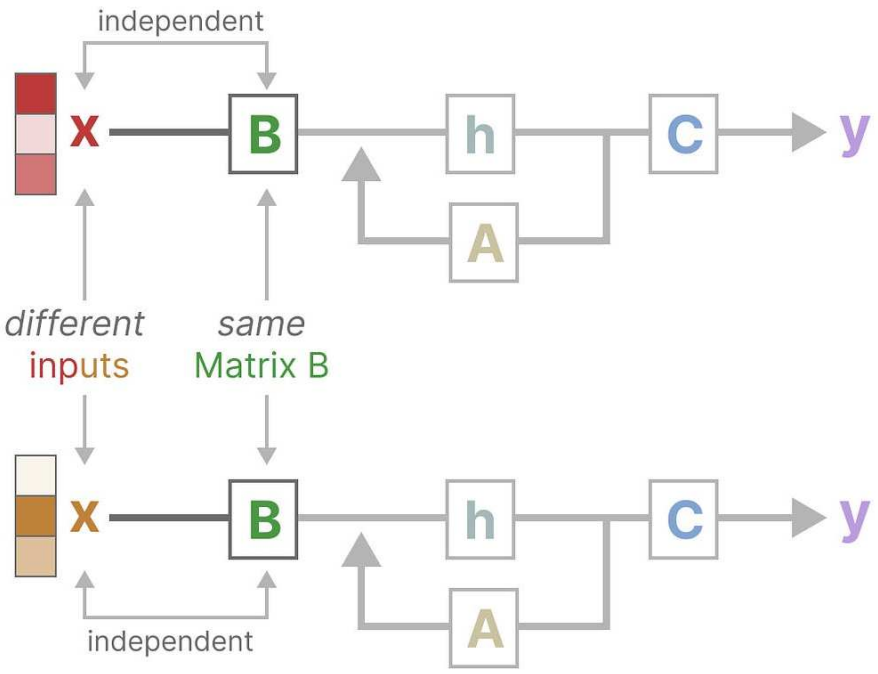


- In the above example, we are essentially performing ***one-shot prompting*** where we attempt to "teach" the model to provide an "**A**:" response after every "**Q**:". However, since SSMs are ***time-invariant*** it cannot select which previous tokens to recall from its history.

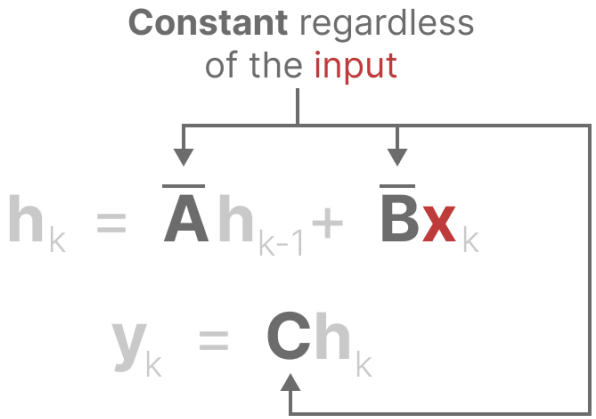
Part 3: Mamba - A Selective SSM

➢ What Problem does it attempt to Solve?

- Let's illustrate this by focusing on ***matrix* B**. Regardless of what the input ***x*** is, ***matrix* B** remains exactly the same and is therefore independent of ***x***:



- Likewise, A and C also remain fixed regardless of the input. This demonstrates the static nature of the SSMs we have seen thus far.

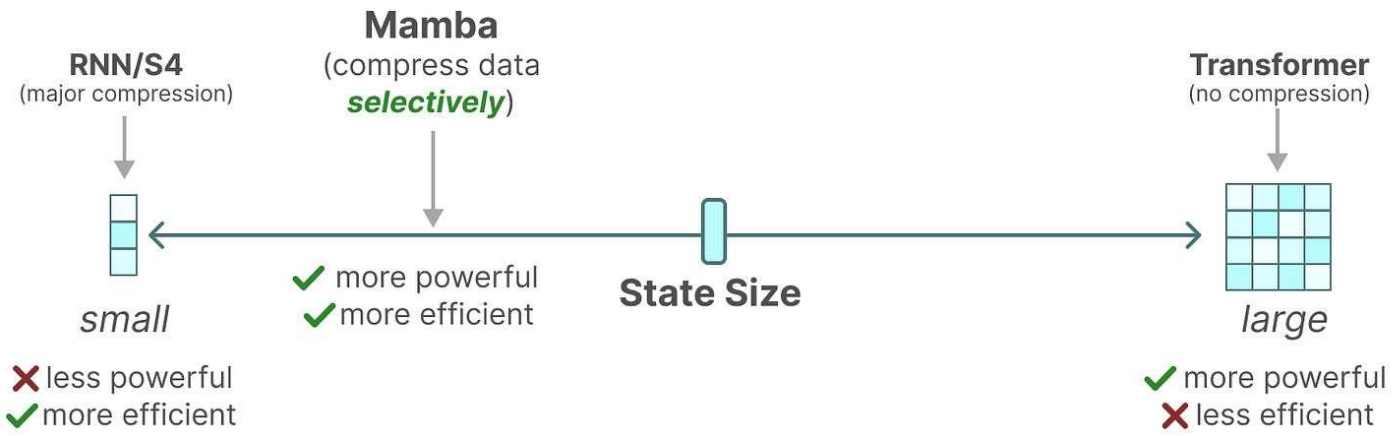


- In comparison, these tasks are relatively easy for **Transformers** since they dynamically change their attention based on the input sequence. They can selectively "look" or "attend" at different parts of the sequence.
- The ***poor*** performance of **SSMs** on these tasks illustrates the underlying problem with ***time-invariant SSMs***, the static nature of *matrices* A, B, and C results in problems with ***content-awareness***.

Part 3: Mamba - A Selective SSM

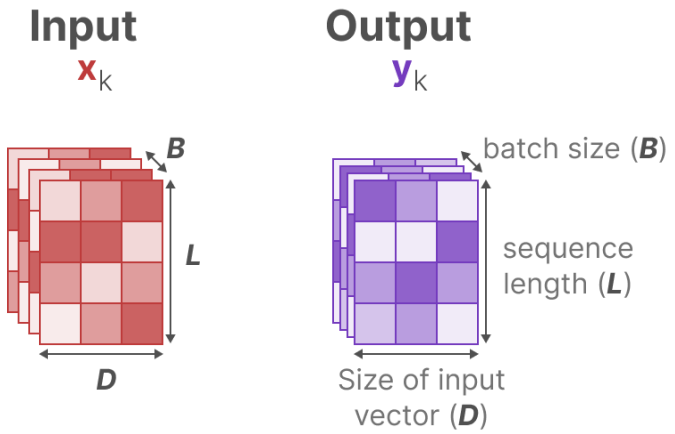
➢ Selectively Retain Information

- The recurrent representation of an SSM creates a small state that is quite efficient as it compresses the entire history. However, compared to a *Transformer* model which does no compression of the history (through the attention matrix), it is much less powerful.
- **Mamba** aims to have the best of both worlds. **A small state that is as powerful as the state of a Transformer**:
- As teased above, it does so by compressing data selectively into the state. When you have an input sentence, there is often information, like stop words, that does not have much meaning.

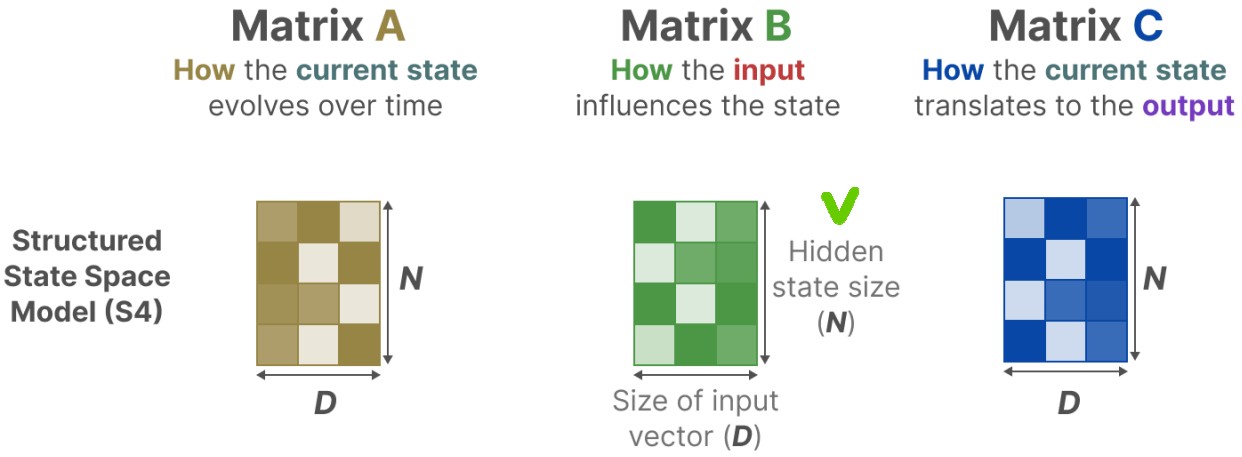


➢ Selectively Retain Information

- *To selectively compress information, we need the parameters to be dependent on the input*.
- To do so, let's first explore the dimensions of the input and output in an SSM during training:



- In a **Structured State Space Model (S4)**, the *matrices* A, B, and C are independent of the input since their dimensions ***N*** and ***D*** are static and do not change.



Part 3: Mamba - A Selective SSM

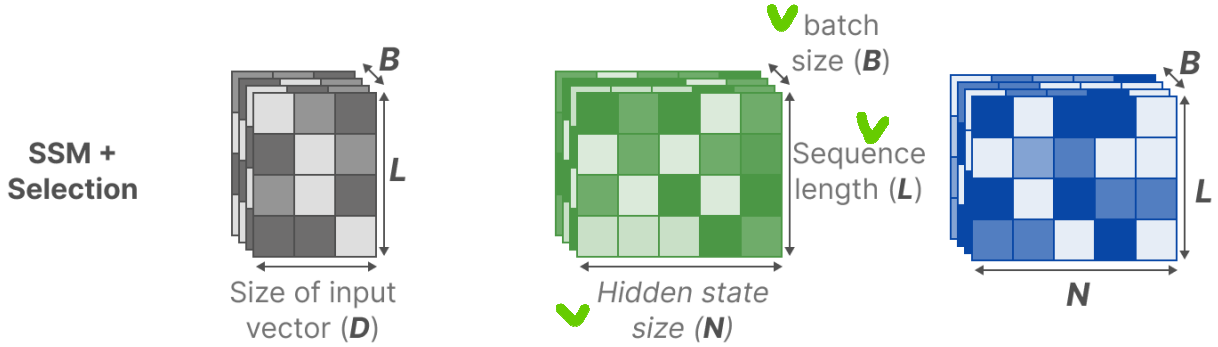
➢ Selectively Retain Information

- Instead, **Mamba** makes *matrices* B and C, and even the *step size* Δ, dependent on the input by incorporating the sequence length and batch size of the input:

Step size (Δ)
Resolution of the **input**
(discretization parameter)

Matrix B
How the **input**
influences the state

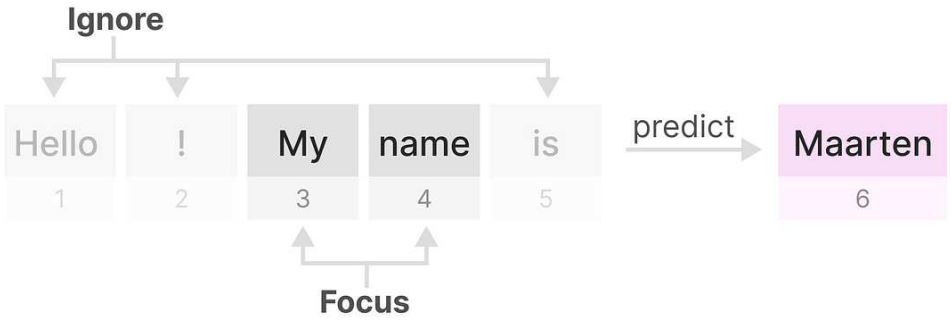
Matrix C
How the **current state**
translates to the **output**



❖ This means that for every input token, we now have different B and C matrices which solves the problem with ***content-awareness***!

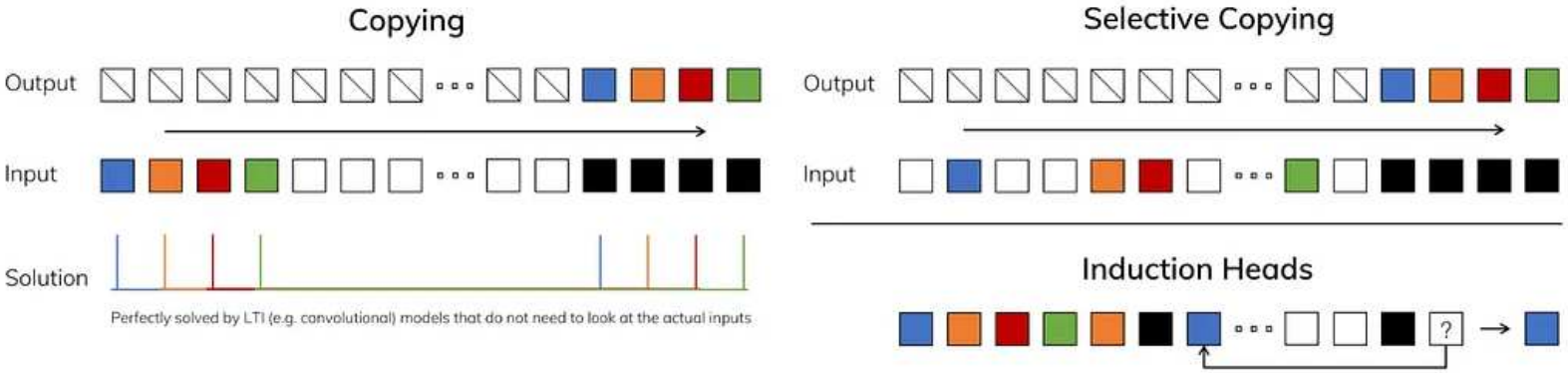
❖ **NOTE**: *Matrix* A remains the same since we want the state itself to remain static but the way it is influenced (through B and C) to be dynamic.

- Together, they *selectively* choose what to keep in the hidden state and what to ignore since they are now dependent on the input.
- A **smaller *step size*** Δ results in ignoring specific words and instead using the previous context more whilst a **larger *step size*** Δ focuses on the input words more than the context:



Part 3: Mamba - A Selective SSM

- **Selectively Retain Information**



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: $\Delta : (\mathtt{D}) \leftarrow \tau_\Delta(\text{Parameter})$
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

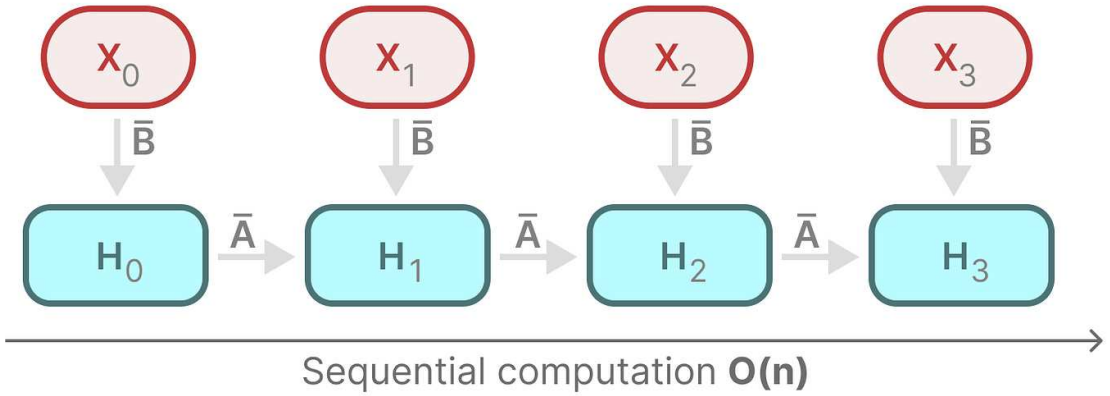
Algorithm 2 SSM + Selection (S6)

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

Part 3: Mamba - A Selective SSM

➢ The Scan Operation

- Since these **matrices** are now ***dynamic***, they cannot be calculated using the convolution representation since it assumes *a fixed kernel*. We can only use the recurrent representation and lose the parallelization the convolution provides.
- To enable ***parallelization***, let's explore how we compute the output with recurrency:

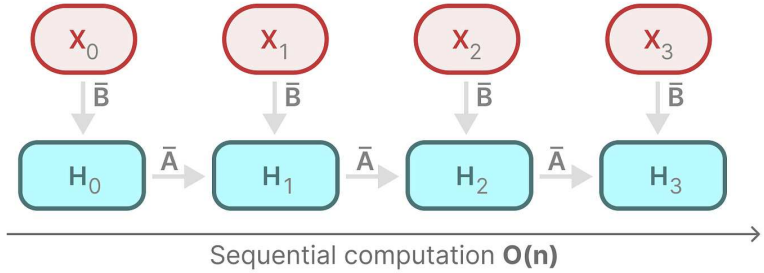


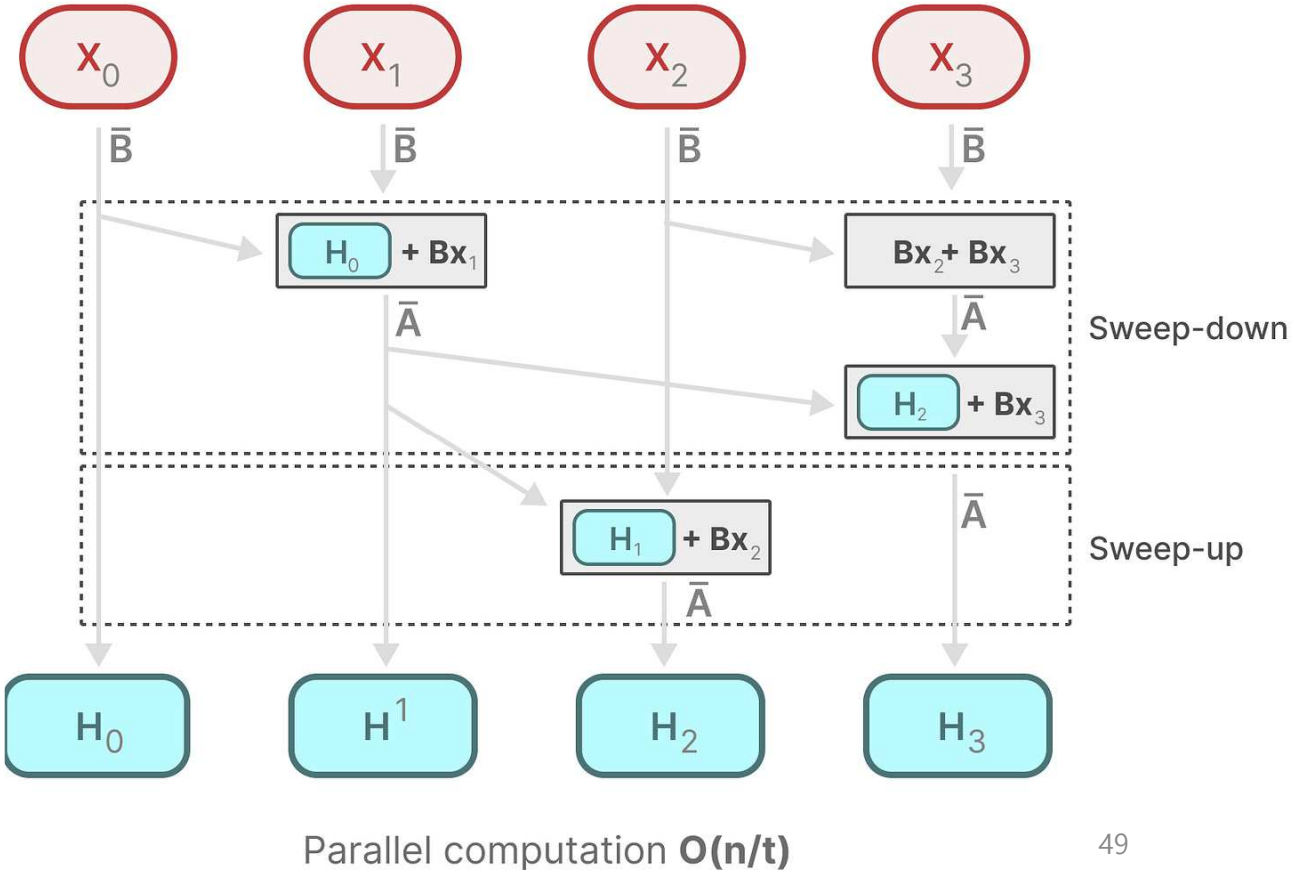
- Each state is the sum of the previous state (multiplied by A) plus the current input (multiplied by B). This is called a ***scan operation*** and can easily be calculated with a for loop.
- Parallelization, in contrast, seems impossible since each state can only be calculated if we have the previous state. **Mamba**, however, **makes this possible through the parallel scan algorithm.**

Part 3: Mamba - A Selective SSM

➢ The Scan Operation

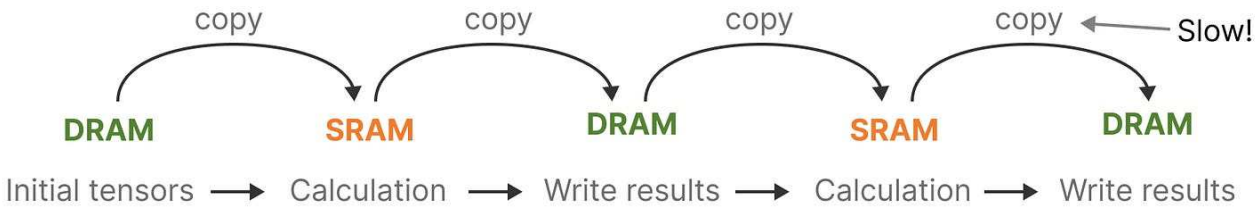
- Parallelization, in contrast, seems impossible since each state can only be calculated if we have the previous state. **Mamba**, however, **makes this possible through the parallel scan algorithm**.
- It assumes the order in which we do operations does not matter through the associate property. As a result, we can calculate the sequences in parts and iteratively combine them:
- Together, *dynamic matrices B and C, and the parallel scan algorithm* create the ***selective scan algorithm*** to represent the ***dynamic*** and ***fast*** nature of using *the recurrent representation*.



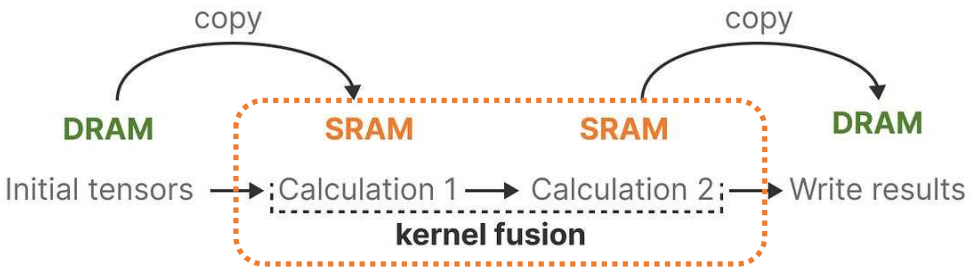


➢ Hardware-aware Algorithm

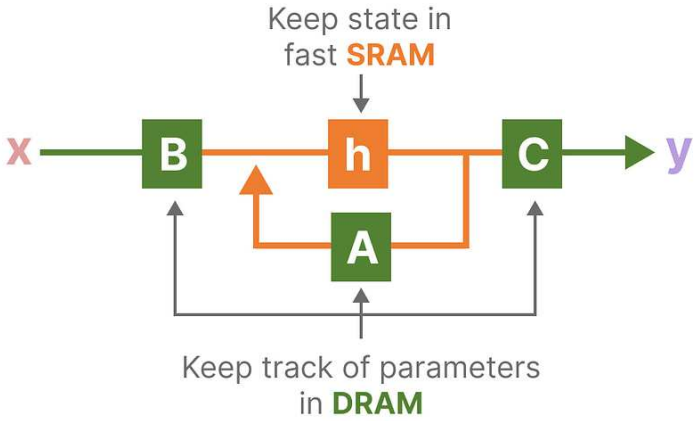
- A disadvantage of recent GPUs is their limited transfer (IO) speed between their small but highly efficient SRAM and their large but slightly less efficient DRAM. Frequently copying information between SRAM and DRAM becomes a bottleneck.



- Mamba, like *Flash Attention*, attempts to limit the number of times we need to go from DRAM to SRAM and vice versa. It does so through ***kernel fusion*** which allows the model to prevent writing intermediate results and continuously performing computations until it is done.



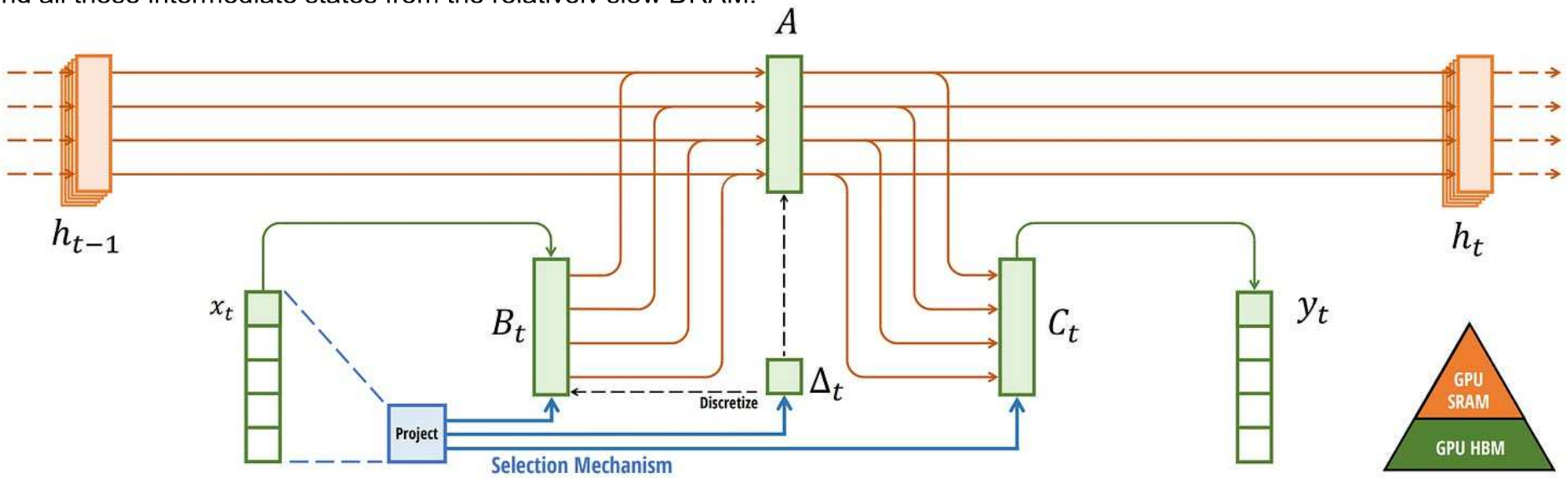
- We can view the specific instances of DRAM and SRAM allocation by visualizing Mamba's base architecture:



- Here, the following are fused into one kernel:
  - ✓ Discretization step with step size Δ
  - ✓ Selective scan algorithm
  - ✓ Multiplication with C

➢ Hardware-aware Algorithm

- The last piece of the hardware-aware algorithm is ***recomputation***.
- The intermediate states are not saved but are necessary for the backward pass to compute the gradients. Instead, *the authors recompute those intermediate states during the backward pass.*
- Although this might seem inefficient, it is much less costly than reading all those intermediate states from the relatively slow DRAM.
- We have now covered all components of its architecture which is depicted using the following image from its article:
- This architecture is often referred to as a ***selective SSM*** or ***S6*** model since it is essentially an S4 model computed with the selective scan algorithm.

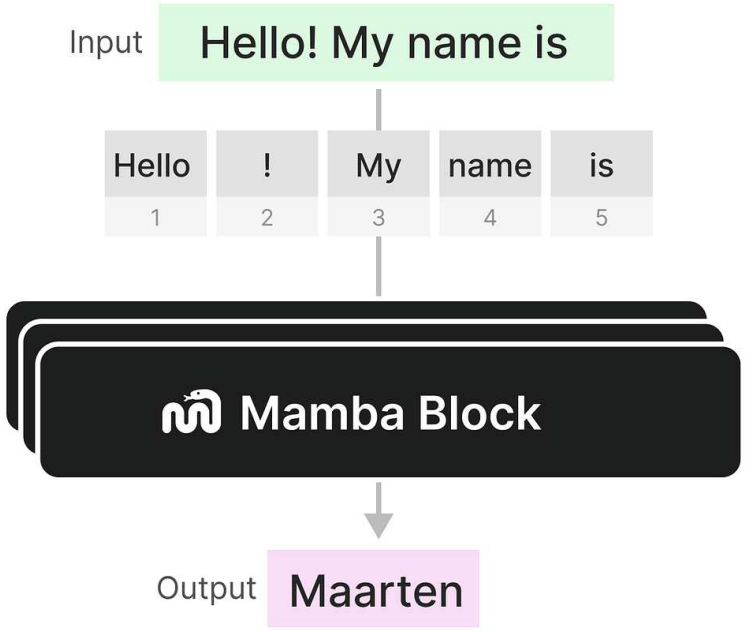


The Selective SSM. Retrieved from: Gu, Albert, and Tri Dao. "Mamba: Linear-time sequence modeling with selective state spaces." arXiv preprint arXiv:2312.00752 (2023).

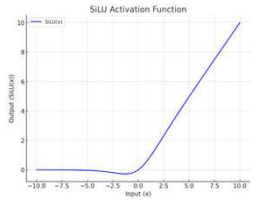
Part 3: Mamba - A Selective SSM

➢ The Mamba Block

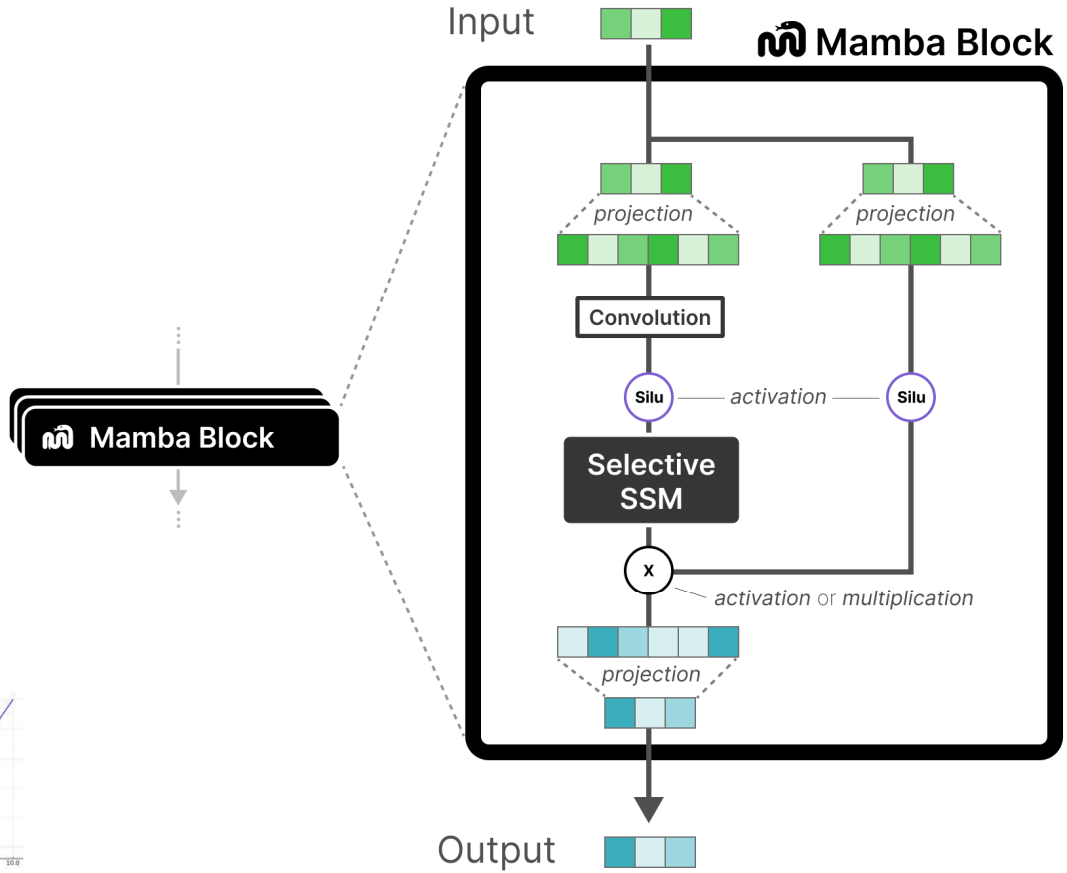
- The ***selective SSM*** that we have explored thus far can be implemented as a *block*, the same way we can represent self-attention in a decoder block.



❖ SiLU (Sigmoid Linear Unit Or Swish) activation function



- Like the decoder, we can stack *multiple Mamba blocks* and use their output as the input for the next Mamba block:

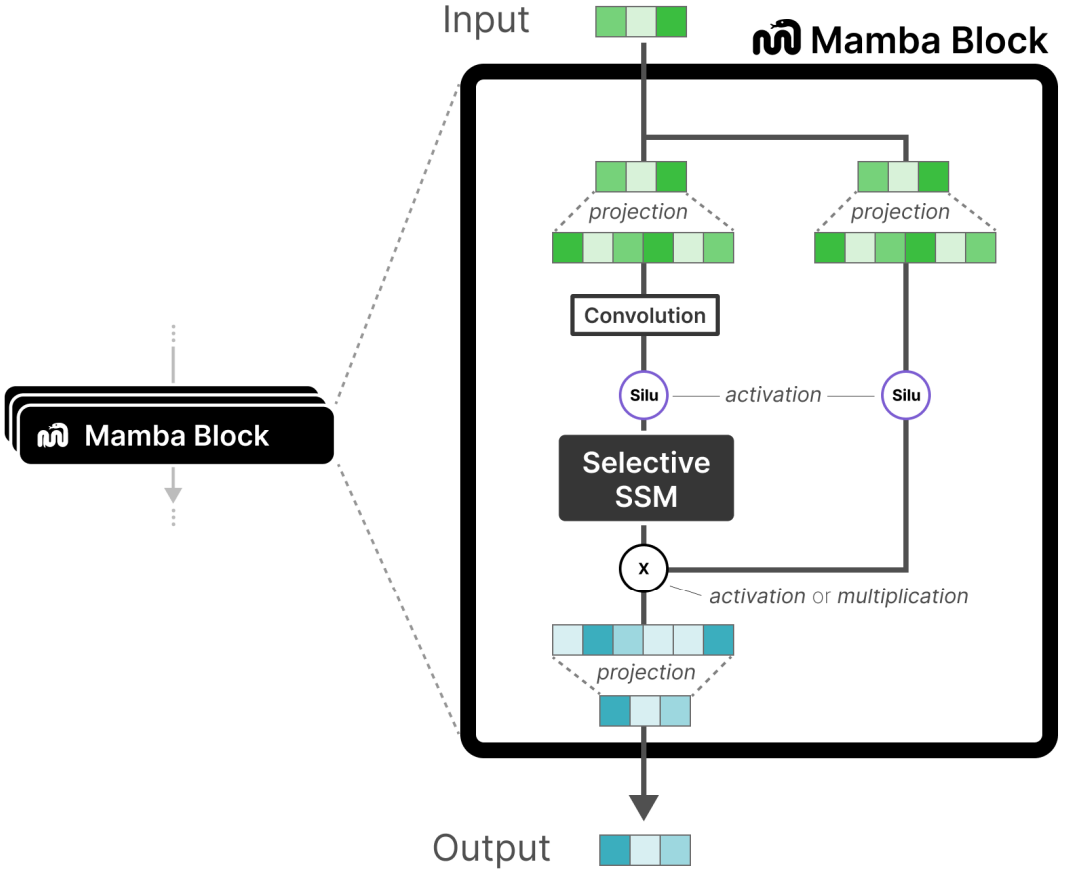


Part 3: Mamba - A Selective SSM

➢ The Mamba Block

- It starts with a *linear projection* to expand upon the input embeddings. Then, a *convolution* before the Selective SSM is applied *to prevent independent token calculations*.

- The **Selective SSM** has the following properties:
  - ✓ Recurrent SSM created through discretization
  - ✓ HiPPO initialization on matrix A to capture long-range dependencies
  - ✓ Selective scan algorithm to selectively compress information
  - ✓ Hardware-aware algorithm to speed up computation



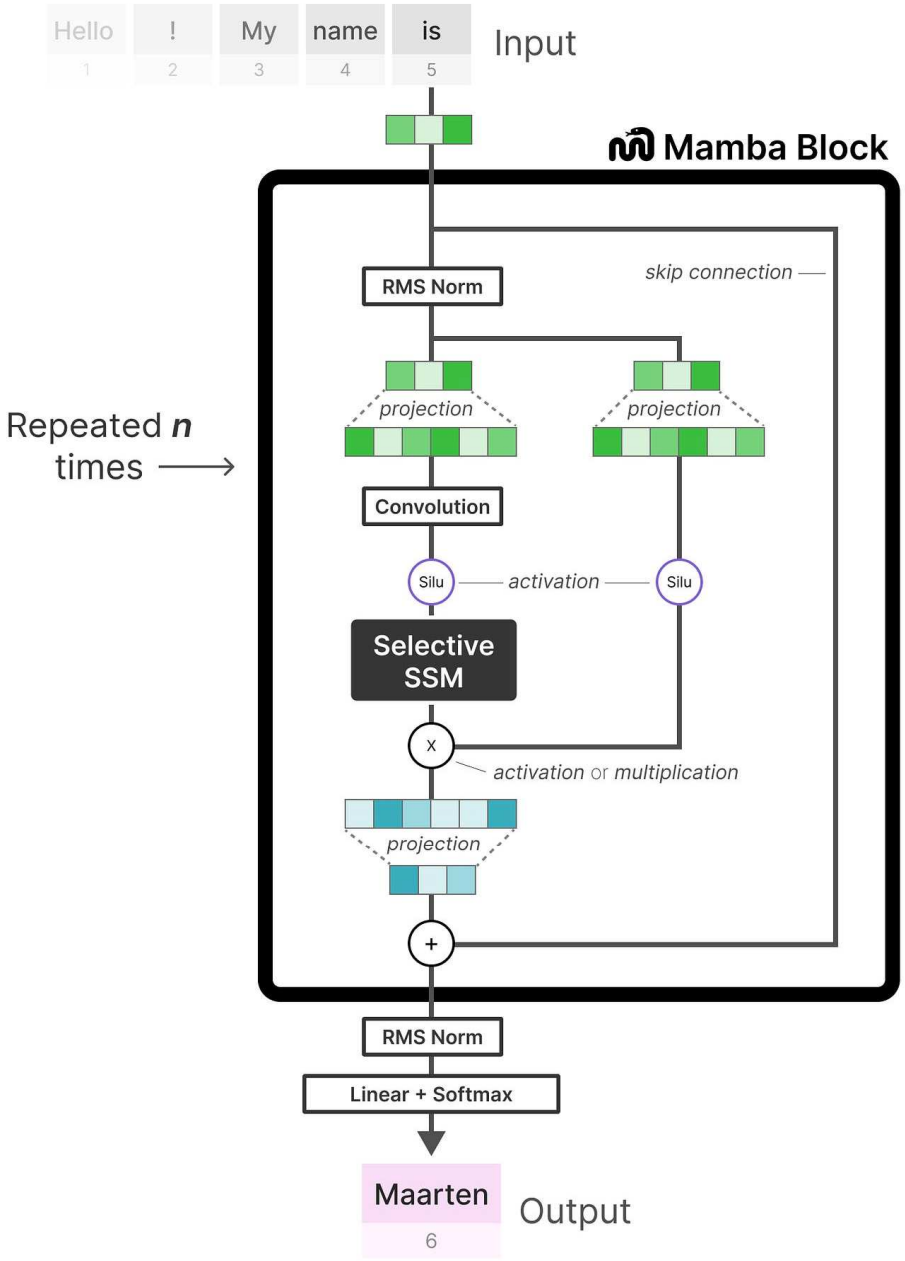
Part 3: Mamba - A Selective SSM

➢ The Mamba Block

- We can expand on this architecture a bit more when looking at the code implementation and explore how an end-to-end example would look like:

- Notice some changes, like the inclusion of normalization layers and softmax for choosing the output token.

- When we put everything together, we get both fast inference and training and even unbounded context. Using this architecture, the authors found it matches and sometimes even exceeds the performance of Transformer models of the same size!



Part 3: Mamba - A Selective SSM

- **The Mamba Block**

	Training	Inference
Transformers	**Fast!** (parallelizable)	**Slow...** (scales **quadratically** with sequence length)
RNNs	**Slow...** (not parallelizable)	**Fast!** (scales **linearly** with sequence length)
Mamba	**Fast!** (parallelizable)	**Fast!** (scales **linearly** with sequence length + **unbounded** context)