SSMs: State Space Models

Prerequisites to Mamba

Plan

- Review: Sequence modelling
- Motivation for the new architecture
- Alternatives to attention:
  - Linear RNNs
  - State space models
    - Mathematical model
    - Discretization
    - Recurrent/Convolutional computation
    - Structure and Dimensions
- Prior work to Mamba: Architectures with SSMs

Review: Sequence modelling

- RNNs
- CNNs
- Transformer

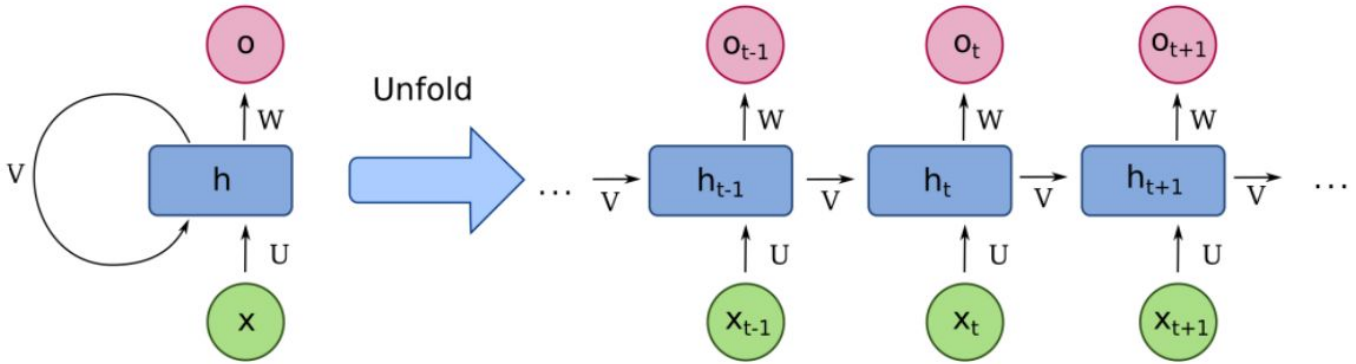
Review: RNNs

Theoretically with infinite context window

Practically suffer from vanishing/exploding gradient

Training: $O(N)$, not parallelizable

Inference: constant time for each token



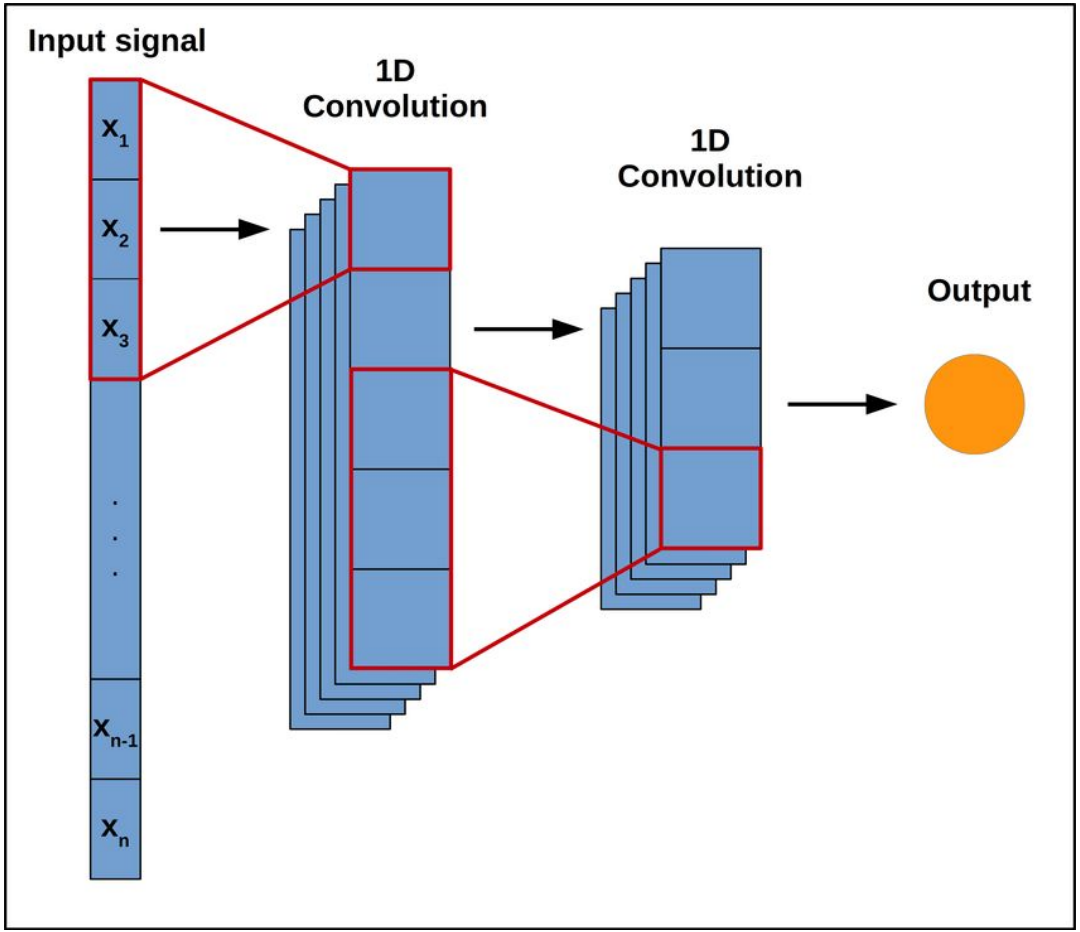
Review: CNNs

Finite context window (depending on kernel size)

Need to materialize the kernel before using it

Training and inference depend on kernel size

Easily parallelizable

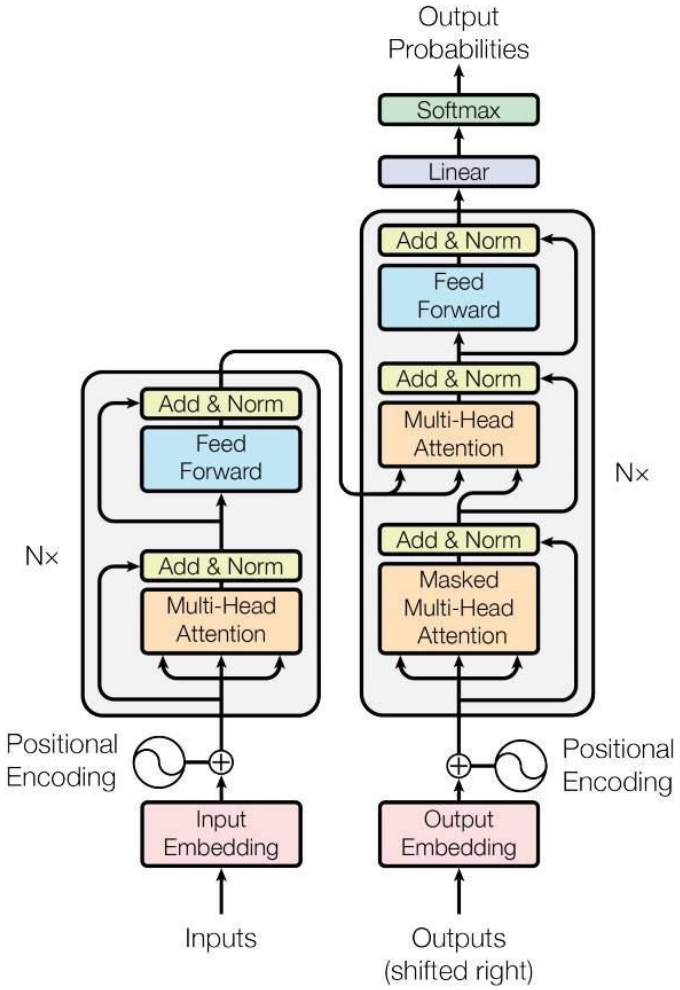


Review: Transformers

Finite context window

Training: $O(N^2)$, easily parallelizable

Inference: $O(N)$ when using KV-Cache, for each token



Motivation for the new architecture

We want something as efficient as RNNs:

1. constant-time per step inference
2. linear training + parallelizable

… and as good at performance as Transformers

Motivation for the new architecture

We want something as efficient as RNNs:

1. constant-time per step inference
2. linear training + parallelizable

… and as good at performance as Transformers

SSMs is one the directions.

Linear RNNs

Let’s get rid of non-linearity, the basic equation of is thus:

$$h_t = Ah_{t-1} + Bx_t$$
$$y_t = Ch_t$$

Source: https://youtu.be/dKJEpOtVgXc?si=rMeIR3zM3PGv5Q_o

Linear RNNs

Let’s get rid of non-linearity, the basic equation of is thus:

$$h_t = Ah_{t-1} + Bx_t$$
$$y_t = Ch_t$$

- Training Speed: Strong (Parallelizable convolution! – **later**)
- Generation Speed: Strong (constant-time per step)

Source: https://youtu.be/dKJEpOtVgXc?si=rMeIR3zM3PGv5Q_o

Linear RNNs

Let’s get rid of non-linearity, the basic equation of is thus:

$$h_t = Ah_{t-1} + Bx_t$$
$$y_t = Ch_t$$

- Training Speed: Strong (Parallelizable convolution! – **later**)
- Generation Speed: Strong (constant-time per step)
- **Accuracy: Extremely Poor... Barely learns.**

Source: https://youtu.be/dKJEpOtVgXc?si=rMeIR3zM3PGv5Q_o

State space models

Unlike Linear RNNs,

- we look at **continuous-time** values
- instead of modelling hidden state in every position, we use **differential equation** to remodel the change in hidden state over time

$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t)$$
$$y(t) = \boldsymbol{C}h(t) + \boldsymbol{D}x(t)$$

Source: https://youtu.be/dKJEpOtVgXc?si=rMeIR3zM3PGv5Q_o

State space models

A state space model allows us to map an input signal $x(t)$ to an output signal $y(t)$ by means of a state representation $h(t)$ as follows:

$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t)$$
$$y(t) = \boldsymbol{C}h(t) + \boldsymbol{D}x(t)$$

State space models

A state space model allows us to map an input signal $x(t)$ to an output signal $y(t)$ by means of a state representation $h(t)$ as follows:

$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t)$$
$$y(t) = \boldsymbol{C}h(t) + \boldsymbol{D}x(t)$$

Note:

$$x(t) \in \mathbb{R} \mapsto y(t) \in \mathbb{R}$$

$$h(t) \in \mathbb{R}^N$$

$$\boldsymbol{A} \in \mathbb{R}^{N \times N}, \boldsymbol{B} \in \mathbb{R}^{N \times 1}, \boldsymbol{C} \in \mathbb{R}^{1 \times N}$$

Source: https://arxiv.org/abs/2312.00752

State space models

A state space model allows us to map an input signal $x(t)$ to an output signal $y(t)$ by means of a state representation $h(t)$ as follows:

$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t)$$
$$y(t) = \boldsymbol{C}h(t) + \boldsymbol{D}x(t)$$

Note:

$$x(t) \in \mathbb{R} \mapsto y(t) \in \mathbb{R}$$

$$h(t) \in \mathbb{R}^N$$

$$\boldsymbol{A} \in \mathbb{R}^{N \times N}, \boldsymbol{B} \in \mathbb{R}^{N \times 1}, \boldsymbol{C} \in \mathbb{R}^{1 \times N}$$

D is set to zeros,
represents skip-connection

Source: https://arxiv.org/abs/2312.00752

State space models

A state space model allows us to map an input signal $x(t)$ to an output signal $y(t)$ by means of a state representation $h(t)$ as follows:

$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t)$$
$$y(t) = \boldsymbol{C}h(t)$$

Note:

$$x(t) \in \mathbb{R} \mapsto y(t) \in \mathbb{R}$$

$$h(t) \in \mathbb{R}^N$$

$$\boldsymbol{A} \in \mathbb{R}^{N \times N}, \boldsymbol{B} \in \mathbb{R}^{N \times 1}, \boldsymbol{C} \in \mathbb{R}^{1 \times N}$$

D is set to zeros,
represents skip-connection

Source: https://arxiv.org/abs/2312.00752

State space models

A state space model allows us to map an input signal $x(t)$ to an output signal $y(t)$ by means of a state representation $h(t)$ as follows:

$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t)$$
$$y(t) = \boldsymbol{C}h(t)$$

This state space model is linear and time invariant.

Source: https://github.com/hkproj/mamba-notes

State space models

A state space model allows us to map an input signal $x(t)$ to an output signal $y(t)$ by means of a state representation $h(t)$ as follows:

$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t)$$
$$y(t) = \boldsymbol{C}h(t)$$

This state space model is linear and time invariant.

To find the output signal $y(t)$ at time t, we first need to find a function $h(t)$ that describes the state of the system for all time steps.

We first need to discretize our system

Source: https://github.com/hkproj/mamba-notes

Simple Discretization Example

Discretize state space model, so that we can calculate the evolution of the state over time by using a **recurrent formulation**:

Source: https://github.com/hkproj/mamba-notes

Simple Discretization Example

Discretize state space model, so that we can calculate the evolution of the state over time by using a **recurrent formulation**:

1. Using the definition of derivative (fix small Δ): $h(t + \Delta) \cong \Delta h'(t) + h(t)$
2. This is the (continuous) state space model: $h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t)$

Source: https://github.com/hkproj/mamba-notes

Simple Discretization Example

Discretize state space model, so that we can calculate the evolution of the state over time by using a **recurrent formulation**:

1. Using the definition of derivative (fix small Δ): $h(t + \Delta) \cong \Delta h'(t) + h(t)$
2. This is the (continuous) state space model: $h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t)$
3. Substitute the state space model into the first expression:

$$
\begin{aligned}
h(t+\Delta) &\cong \Delta\big(\boldsymbol{A}h(t) + \boldsymbol{B}x(t)\big) + h(t) \\
&= \Delta\boldsymbol{A}h(t) + \Delta\boldsymbol{B}x(t) + h(t) \\
&= (I + \Delta\boldsymbol{A})h(t) + \Delta\boldsymbol{B}x(t) \\
&= \overline{\boldsymbol{A}}h(t) + \overline{\boldsymbol{B}}x(t)
\end{aligned}
$$

Source: https://github.com/hkproj/mamba-notes

Mamba Discretization

The matrices $\overline{\boldsymbol{A}}$ and $\overline{\boldsymbol{B}}$ are the discretized parameters of the model.

This allows us to also calculate the output $y(t)$ of the system for discrete time steps.

$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t) \quad (1a) \qquad h_t = \overline{\boldsymbol{A}}h_{t-1} + \overline{\boldsymbol{B}}x_t \quad (2a)$$

$$y(t) = \boldsymbol{C}h(t) \quad (1b) \qquad y_t = \boldsymbol{C}h_t \quad (2b)$$

Source: https://github.com/hkproj/mamba-notes, https://arxiv.org/abs/2312.00752

Mamba Discretization

The matrices $\overline{\boldsymbol{A}}$ and $\overline{\boldsymbol{B}}$ are the discretized parameters of the model.

This allows us to also calculate the output $y(t)$ of the system for discrete time steps.

$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t) \quad (1a) \qquad h_t = \overline{\boldsymbol{A}}h_{t-1} + \overline{\boldsymbol{B}}x_t \quad (2a)$$

$$y(t) = \boldsymbol{C}h(t) \quad (1b) \qquad y_t = \boldsymbol{C}h_t \quad (2b)$$

For **Mamba**, **Zero-Order Hold (ZOH)** rule is used to discretize the system:

Discretization. The first stage transforms the "continuous parameters" $(\Delta, \boldsymbol{A}, \boldsymbol{B})$ to "discrete parameters" $(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}})$ through fixed formulas $\overline{\boldsymbol{A}} = f_A(\Delta, \boldsymbol{A})$ and $\overline{\boldsymbol{B}} = f_B(\Delta, \boldsymbol{A}, \boldsymbol{B})$, where the pair (f_A, f_B) is called a discretization rule. Various rules can be used such as the zero-order hold (ZOH) defined in equation (4).

$$\overline{\boldsymbol{A}} = \exp(\Delta \boldsymbol{A}) \qquad \overline{\boldsymbol{B}} = (\Delta \boldsymbol{A})^{-1}(\exp(\Delta \boldsymbol{A}) - \boldsymbol{I}) \cdot \Delta \boldsymbol{B} \quad (4)$$

Source: https://github.com/hkproj/mamba-notes, https://arxiv.org/abs/2312.00752

Mamba Discretization

$$h'(t) = \boldsymbol{A}h(t) + \boldsymbol{B}x(t) \quad (1a) \qquad h_t = \overline{\boldsymbol{A}}h_{t-1} + \overline{\boldsymbol{B}}x_t \quad (2a)$$

$$y(t) = \boldsymbol{C}h(t) \quad (1b) \qquad y_t = \boldsymbol{C}h_t \quad (2b)$$

For **Mamba**, **Zero-Order Hold (ZOH)** rule is used to discretize the system:

Discretization. The first stage transforms the "continuous parameters" $(\Delta, \boldsymbol{A}, \boldsymbol{B})$ to "discrete parameters" $(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}})$ through fixed formulas $\overline{\boldsymbol{A}} = f_A(\Delta, \boldsymbol{A})$ and $\overline{\boldsymbol{B}} = f_B(\Delta, \boldsymbol{A}, \boldsymbol{B})$, where the pair (f_A, f_B) is called a discretization rule. Various rules can be used such as the zero-order hold (ZOH) defined in equation (4).

$$\overline{\boldsymbol{A}} = \exp(\Delta \boldsymbol{A}) \qquad \overline{\boldsymbol{B}} = (\Delta \boldsymbol{A})^{-1}(\exp(\Delta \boldsymbol{A}) - \boldsymbol{I}) \cdot \Delta \boldsymbol{B} \quad (4)$$

Note: Δ is learnable parameter

Source: https://github.com/hkproj/mamba-notes, https://arxiv.org/abs/2312.00752

Recurrent computation mode

Discretization leads to recurrent formulation.

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

Suppose that, for simplicity, the initial state of the system is 0.

$$h_0 = \bar{B}x_0$$
$$y_0 = Ch_0$$

$$h_1 = \bar{A}h_0 + \bar{B}x_1$$
$$y_1 = Ch_1$$

$$h_2 = \bar{A}h_1 + \bar{B}x_2$$
$$y_2 = Ch_2$$

Source: https://github.com/hkproj/mamba-notes, https://arxiv.org/abs/2312.00752

Recurrent computation: the problem

The recurrent formulation is great for inference: $O(1)$

The recurrent formulation is not good for training: $O(N)$

We want to parallelize the computation as much as possible, just like the Transformer does.

Source: https://github.com/hkproj/mamba-notes, https://arxiv.org/abs/2312.00752

Recurrent computation: the problem

The recurrent formulation is great for inference: $O(1)$

The recurrent formulation is not good for training: $O(N)$

We want to parallelize the computation as much as possible, just like the Transformer does.

Thankfully, State Space Models provide a convolutional mode as well, let's see how it works!

Source: https://github.com/hkproj/mamba-notes, https://arxiv.org/abs/2312.00752

Convolutional computation mode

$$h_0 = \bar{B}x_0$$
$$y_0 = Ch_0 = C\bar{B}x_0$$

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

Source: https://github.com/hkproj/mamba-notes, https://arxiv.org/abs/2312.00752

Convolutional computation mode

$$h_0 = \bar{B}x_0$$
$$y_0 = Ch_0 = C\bar{B}x_0$$

$$h_1 = \bar{A}h_0 + \bar{B}x_1 = \bar{A}\bar{B}x_0 + \bar{B}x_1$$
$$y_1 = Ch_1 = C(\bar{A}\bar{B}x_0 + \bar{B}x_1) = C\bar{A}\bar{B}x_0 + C\bar{B}x_1$$

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

Source: https://github.com/hkproj/mamba-notes, https://arxiv.org/abs/2312.00752

Convolutional computation mode

$$h_0 = \bar{B}x_0$$
$$y_0 = Ch_0 = C\bar{B}x_0$$

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

$$h_1 = \bar{A}h_0 + \bar{B}x_1 = \bar{A}\bar{B}x_0 + \bar{B}x_1$$
$$y_1 = Ch_1 = C(\bar{A}\bar{B}x_0 + \bar{B}x_1) = C\bar{A}\bar{B}x_0 + C\bar{B}x_1$$

$$h_2 = \bar{A}h_1 + \bar{B}x_2 = \bar{A}(\bar{A}\bar{B}x_0 + \bar{B}x_1) + \bar{B}x_2 = \bar{A}^2\bar{B}x_0 + \bar{A}\bar{B}x_1 + \bar{B}x_2$$
$$y_2 = Ch_2 = C(\bar{A}^2\bar{B}x_0 + \bar{A}\bar{B}x_1 + \bar{B}x_2) = C\bar{A}^2\bar{B}x_0 + C\bar{A}\bar{B}x_1 + C\bar{B}x_2$$

Source: https://github.com/hkproj/mamba-notes, https://arxiv.org/abs/2312.00752

Convolutional computation mode

$$h_0 = \bar{B}x_0$$
$$y_0 = Ch_0 = C\bar{B}x_0$$

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

$$h_1 = \bar{A}h_0 + \bar{B}x_1 = \bar{A}\bar{B}x_0 + \bar{B}x_1$$
$$y_1 = Ch_1 = C(\bar{A}\bar{B}x_0 + \bar{B}x_1) = C\bar{A}\bar{B}x_0 + C\bar{B}x_1$$

$$h_2 = \bar{A}h_1 + \bar{B}x_2 = \bar{A}(\bar{A}\bar{B}x_0 + \bar{B}x_1) + \bar{B}x_2 = \bar{A}^2\bar{B}x_0 + \bar{A}\bar{B}x_1 + \bar{B}x_2$$
$$y_2 = Ch_2 = C(\bar{A}^2\bar{B}x_0 + \bar{A}\bar{B}x_1 + \bar{B}x_2) = C\bar{A}^2\bar{B}x_0 + C\bar{A}\bar{B}x_1 + C\bar{B}x_2$$

$$y_k = C\bar{A}^k\bar{B}x_0 + C\bar{A}^{k-1}\bar{B}x_1 + \cdots + C\bar{A}\bar{B}x_{k-1} + C\bar{B}x_k$$

Source: https://github.com/hkproj/mamba-notes, https://arxiv.org/abs/2312.00752

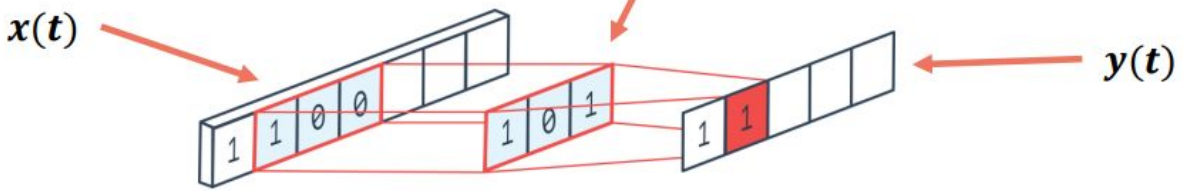
Convolutional computation mode

The output of the system can be calculated using a convolution of a kernel K with the input $x(t)$.

$$y_k = C\bar{A}^k\bar{B}x_0 + C\bar{A}^{k-1}\bar{B}x_1 + \cdots + C\bar{A}\bar{B}x_{k-1} + C\bar{B}x_k$$

$$\overline{\boldsymbol{K}} = (C\overline{\boldsymbol{B}}, C\overline{\boldsymbol{AB}}, \ldots, C\overline{\boldsymbol{A}}^{\boldsymbol{k}}\overline{\boldsymbol{B}}, \ldots) \quad (3a)$$

$$y = x * \overline{\boldsymbol{K}} \quad (3b)$$



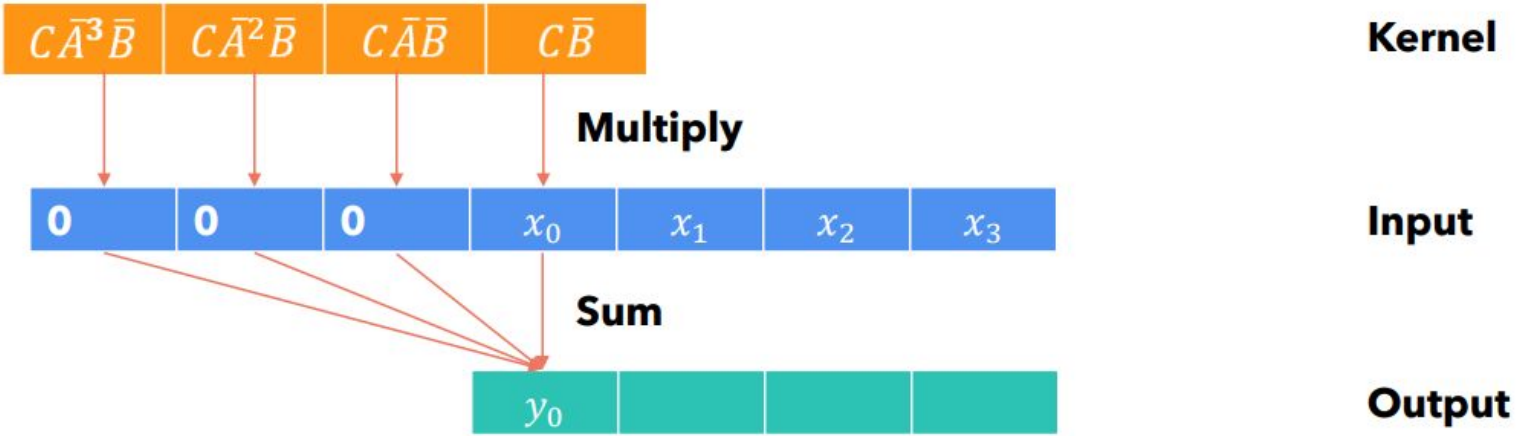
Source: https://github.com/hkproj/mamba-notes, https://arxiv.org/abs/2312.00752

Convolutional computation: step 0

$$y_k = C\bar{A}^k\bar{B}x_0 + C\bar{A}^{k-1}\bar{B}x_1 + \cdots + C\bar{A}\bar{B}x_{k-1} + C\bar{B}x_k$$

$$\overline{\boldsymbol{K}} = (C\overline{\boldsymbol{B}}, C\overline{\boldsymbol{AB}}, \ldots, C\overline{\boldsymbol{A}}^{\boldsymbol{k}}\overline{\boldsymbol{B}}, \ldots) \quad (3a)$$

$$y = x * \overline{\boldsymbol{K}} \quad (3b)$$



$$y_0 = C\bar{B}x_0$$

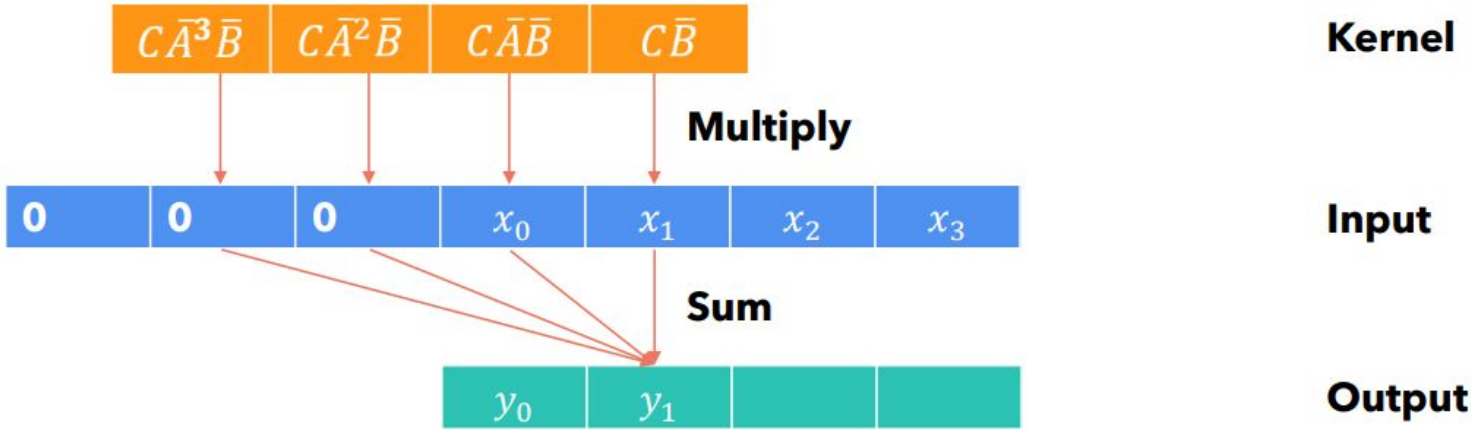
Source: https://github.com/hkproj/mamba-notes, https://arxiv.org/abs/2312.00752

Convolutional computation: step 1

$$y_k = C\bar{A}^k\bar{B}x_0 + C\bar{A}^{k-1}\bar{B}x_1 + \cdots + C\bar{A}\bar{B}x_{k-1} + C\bar{B}x_k$$

$$\overline{\boldsymbol{K}} = (C\overline{\boldsymbol{B}}, C\overline{\boldsymbol{AB}}, \ldots, C\overline{\boldsymbol{A}}^{\boldsymbol{k}}\overline{\boldsymbol{B}}, \ldots) \quad (3a)$$

$$y = x * \overline{\boldsymbol{K}} \quad (3b)$$



$$y_1 = C\bar{A}\bar{B}x_0 + C\bar{B}x_1$$

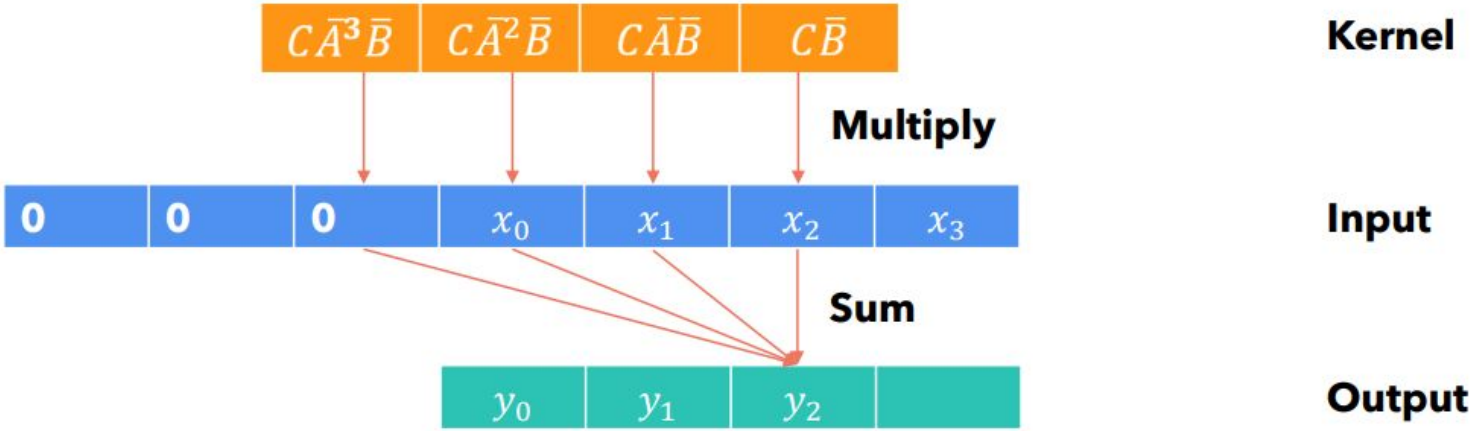
Source: https://github.com/hkproj/mamba-notes, https://arxiv.org/abs/2312.00752

Convolutional computation: step 2

$$y_k = C\bar{A}^k\bar{B}x_0 + C\bar{A}^{k-1}\bar{B}x_1 + \cdots + C\bar{A}\bar{B}x_{k-1} + C\bar{B}x_k$$

$$\overline{\boldsymbol{K}} = (C\overline{\boldsymbol{B}}, C\overline{\boldsymbol{AB}}, \ldots, C\overline{\boldsymbol{A}}^{k}\overline{\boldsymbol{B}}, \ldots) \quad (3a)$$

$$y = x * \overline{\boldsymbol{K}} \quad (3b)$$



$$y_2 = C\bar{A}^2\bar{B}x_0 + C\bar{A}\bar{B}x_1 + C\bar{B}x_2$$

Source: https://github.com/hkproj/mamba-notes, https://arxiv.org/abs/2312.00752

Recurrent/Convolutional mode

The best thing about the convolutional calculation is that it can be parallelized.

Source: https://github.com/hkproj/mamba-notes, https://arxiv.org/abs/2312.00752

Recurrent/Convolutional mode

The best thing about the convolutional calculation is that it can be parallelized.

However, **building the kernel can be computationally expensive**, also from a memory point of view.

Source: https://github.com/hkproj/mamba-notes, https://arxiv.org/abs/2312.00752

Recurrent/Convolutional mode

The best thing about the convolutional calculation is that it can be parallelized.

However, **building the kernel can be computationally expensive**, also from a memory point of view.

1. **Convolutional** mode can be used to perform **training**: we already have all the input sequence of tokens, and it can be easily parallelized.
2. **Recurrent** mode can be used to perform **inference**: one token at a time, using a constant amount of computation and memory for each token.

Source: https://github.com/hkproj/mamba-notes, https://arxiv.org/abs/2312.00752

Structure and Dimensions

$$x(t) \in \mathbb{R} \mapsto y(t) \in \mathbb{R}$$

$$h(t) \in \mathbb{R}^N$$

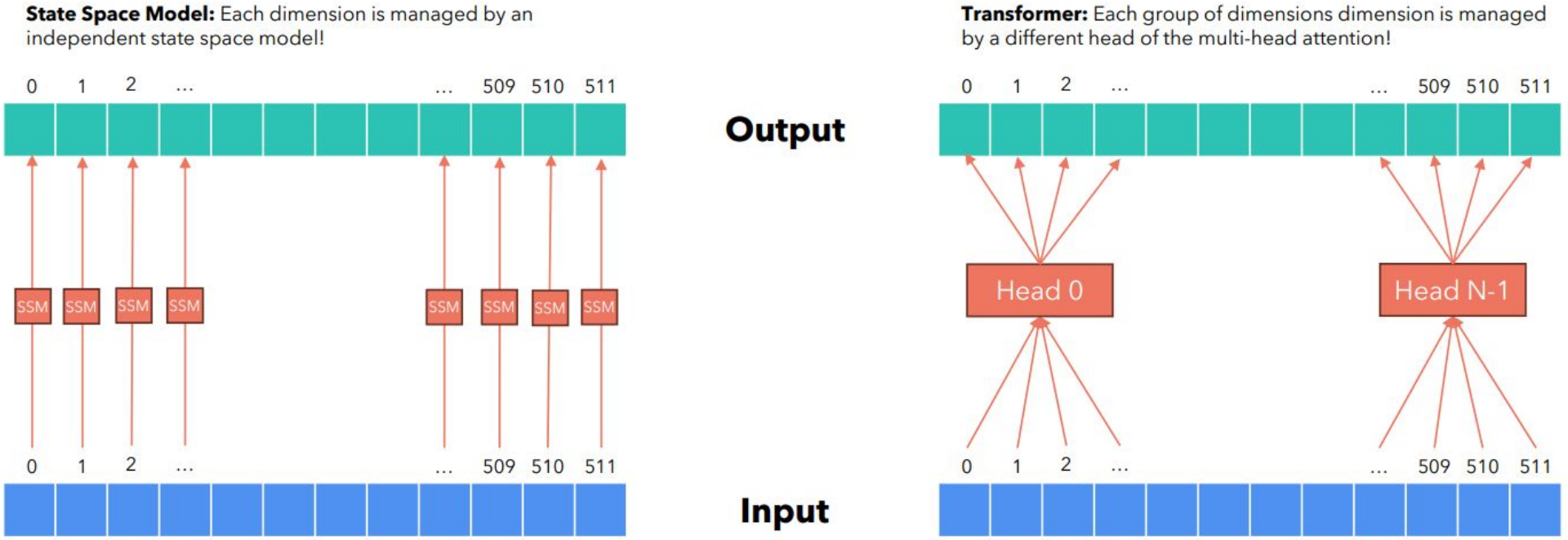
$$\boldsymbol{A} \in \mathbb{R}^{N \times N}, \boldsymbol{B} \in \mathbb{R}^{N \times 1}, \boldsymbol{C} \in \mathbb{R}^{1 \times N}$$

Finally, we note that structured SSMs are so named because computing them efficiently also requires imposing structure on the **A** matrix. The most popular form of structure is **diagonal** which we also use.

In this case, the **matrices can all be represented by N numbers**.

Source: https://arxiv.org/abs/2312.00752

Multiple Dimensions



Source: https://github.com/hkproj/mamba-notes

SSM Algorithm

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

Source: https://arxiv.org/abs/2312.00752

SSM Architectures

Recent research has made significant progress.

S4 [Gu et al., 2022a]
DSS [Gupta, 2022]
GSS [Mehta et al., 2022]
S4D [Gu et al., 2022b]
H3 [Dao et al., 2022]
S5 [Smith et al., 2022]
BiGS [Wang et al., 2022]

QRNN [Bradbury et al., 2016]
Mega [Ma et al., 2022]
SGConv [Li et al., 2022]
Hyena [Poli et al., 2023]
LRU [Orvieto et al., 2023]
RWKV [Peng et al., 2023]
MultiRes [Shi et al., 2023]

Source: https://youtu.be/dKJEpOtVgXc?si=rMeIR3zM3PGv5Q_o

Mamba authors

Tri Dao

Tri Dao
@tri_dao

FlashAttention [Dao et al., 2022]

SSMs research:
H3 [Dao et al., 2022]
Hyena [Poli et al., 2023]

Albert Gu

Albert Gu
@_albertgu

SSMs research:
S4 [Gu et al., 2022a]
DSS [Gupta, 2022]
S4D [Gu et al., 2022b]
BiGS [Wang et al., 2022]
LRU [Orvieto et al., 2023]

Sources

1. Mamba: Linear-Time Sequence Modeling with Selective State Spaces – https://arxiv.org/abs/2312.00752
2. Mamba and S4 Explained: Architecture, Parallel Scan, Kernel Fusion, Recurrent, Convolution, Math – Youtube, Slides
3. Do we need Attention? - Linear RNNs and State Space Models (SSMs) for NLP – Youtube, Slides
4. RNNs strike back – https://adrian-valente.github.io/2023/10/03/linear-rnns.html#rnns-the-new-generation