

# Mamba

## 目录

- [1 Background](#)
  - [1.1 Transformer & Attention Mechanism](#)
  - [1.2 RNN](#)
- [2 From State Space Model to S4 and S6](#)
  - [2.1 SSM](#)
    - [2.1.1 State Space](#)
    - [2.1.2 State Space Model \(SSM\)](#)
  - [2.2 SSM to S4](#)
    - [2.2.1 数据离散化](#)
    - [2.2.2 循环结构表示The Recurrent Representation](#)
    - [2.2.3 卷积结构表示The Convolution Representation](#)
    - [2.2.4 长距离依赖问题: HiPPO](#)
- [3 Mamba: Linear-Time Sequence Modeling with Selective State Spaces](#)
  - [3.1 Mamba = Selection Mechanism + Hardware-aware Algorithm + SSM](#)
    - [3.1.1 From S4 to S6](#)
      - 回顾
      - 关键:
      - 在S4到S6的过程中:
    - [3.1.2 并行扫描\(parallel scan\): 使得不用CNN也能并行训练](#)
    - [3.1.3 硬件感知的状态扩展: 借鉴Flash Attention](#)
    - [3.1.4 简化的SSM架构及最终的整体流程](#)
  - [3.2 mamba的应用实例与一般性的实验结果](#)
    - [3.2.1 通过mamba预测下一个token的示例](#)
    - [3.2.2 三个任务的对比: coping、selective copying、induction heads](#)
    - [3.2.3 实验结果](#)

## Mamba: Linear-Time Sequence Modeling with Selective State Spaces

Albert Gu<sup>\*1</sup> and Tri Dao<sup>\*2</sup>

<sup>1</sup>Machine Learning Department, Carnegie Mellon University

<sup>2</sup>Department of Computer Science, Princeton University

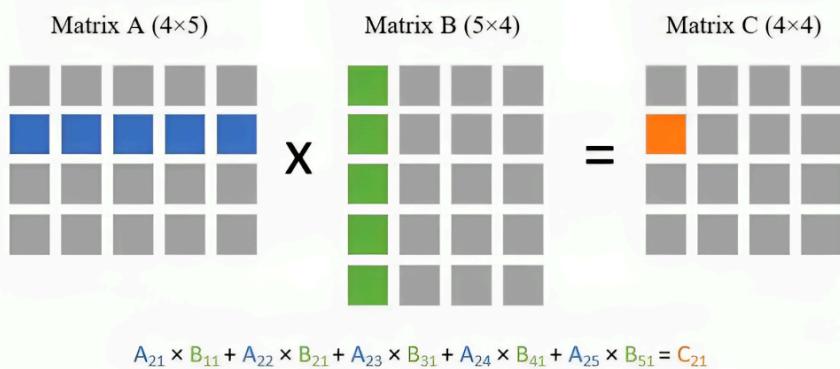
agu@cs.cmu.edu, tri@tridao.me

## 1 Background

### 1.1 Transformer & Attention Mechanism

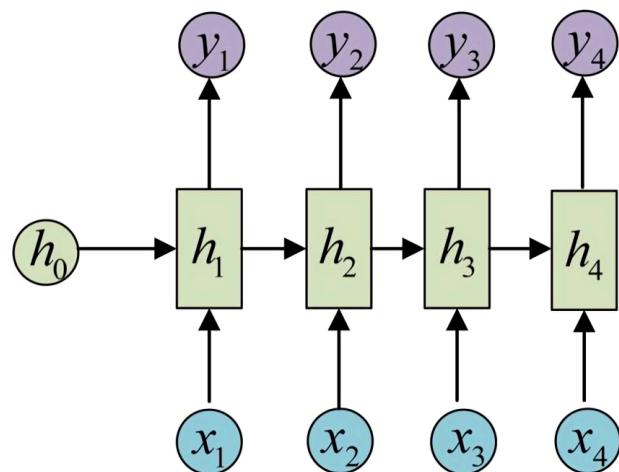
$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

从Self-Attention计算公式来看，它实际上是进行了两个矩阵的点乘。比如两个相乘的矩阵大小分别为( $N \times d$ )和( $d \times N$ )，时间复杂度为： $O(N^2d)$



- 对于越长的文本，模型计算注意力分数所需的资源越多
- 训练速度快，但推理速度慢
- 位置编码
- 长度固定
- ...

## 1.2 RNN

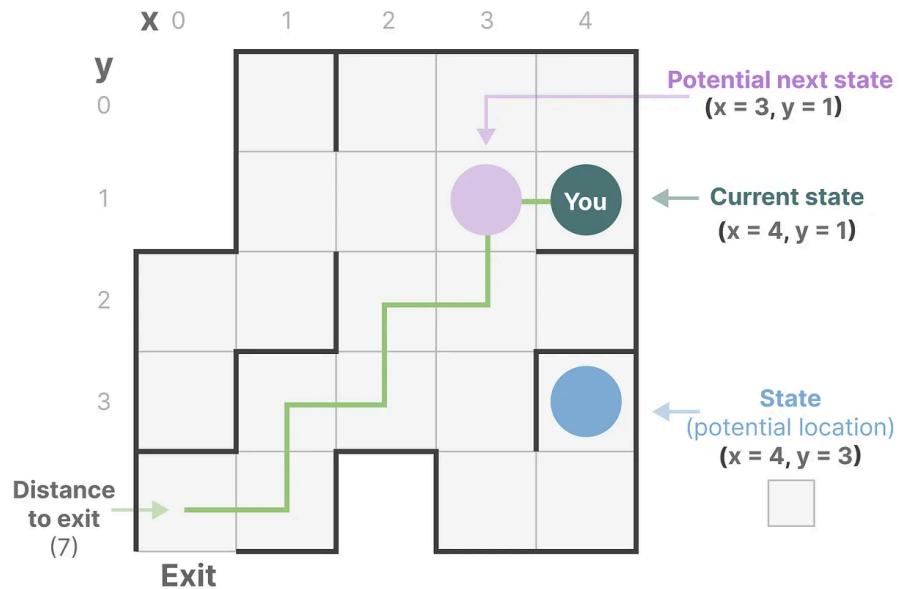


- 无法并行训练
- 虽然每个隐藏状态都是所有先前隐藏状态的聚合，然而随着时间的推移，RNN 往往会忘记某一部分信息

## 2 From State Space Model to S4 and S6

### 2.1 SSM

#### 2.1.1 State Space

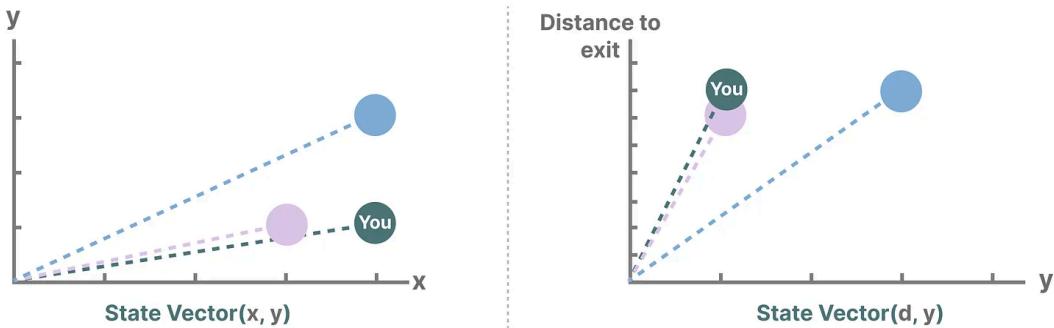


## State Space Representation

每一个小框显示

- 你当前所在的位置(当前状态current state)
- 下一步可以去哪里(未来可能的状态possible future states)
- 以及哪些变化会将你带到下一个状态(向右或向左)

而描述状态的变量(在我们的示例中为 X 和 Y 坐标以及到出口的距离)可以表示为“状态向量state vectors”



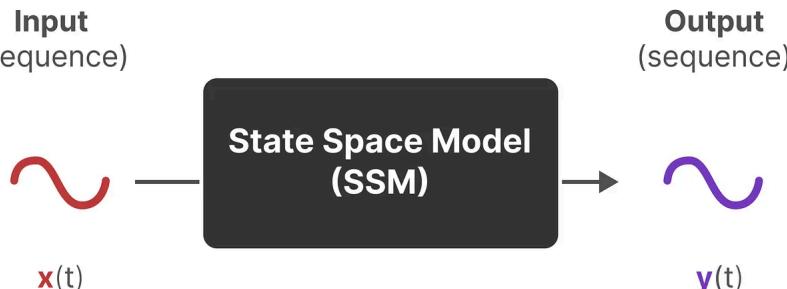
## 2.1.2 State Space Model (SSM)

SSM 是用于描述这些状态表示并根据某些输入预测其下一个状态可能是什么的模型

一般SSMs包括以下组成:

- 映射输入序列  $x(t)$  , 比如在迷宫中向左和向下移动
- 到潜在状态表示  $h(t)$  , 比如距离出口距离和  $x / y$  坐标
- 并导出预测输出序列  $y(t)$  , 比如再次向左移动以更快到达出口

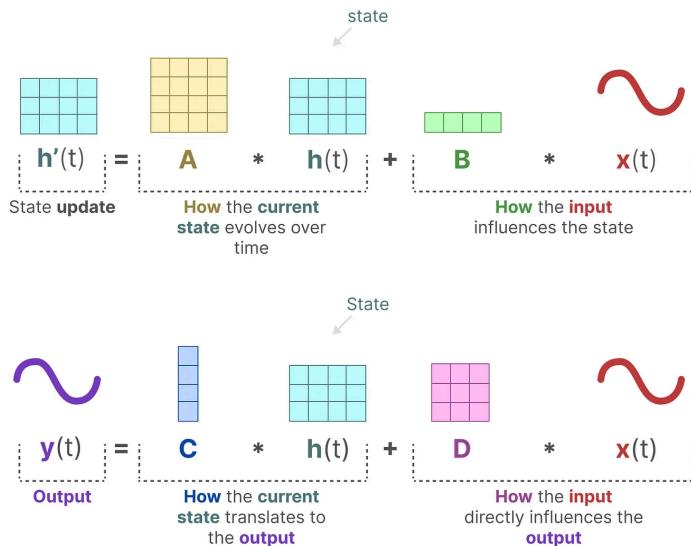
然而, 它不使用离散序列(如向左移动一次), 而是将连续序列作为输入并预测输出序列



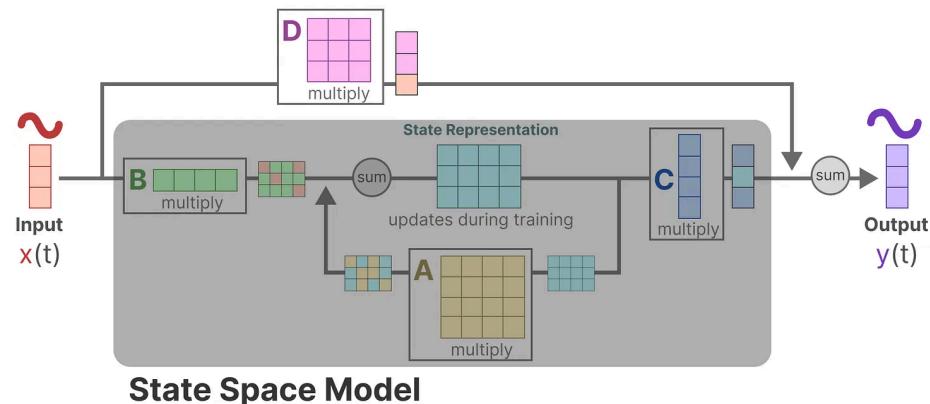
SSM 假设系统(例如在 3D 空间中移动的物体)可以通过两个方程从其在时间  $t$  时的状态进行预测:

- State Equation:  $h(t) = A * h(t - 1) + B * x(t)$
- Output Equation:  $y(t) = C * h(t) + D * x(t)$

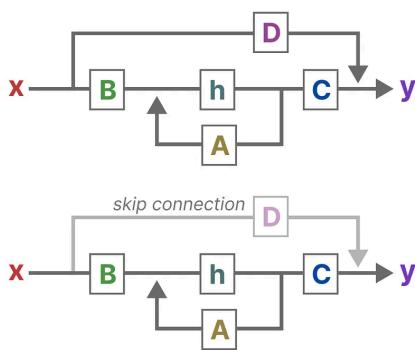
通过求解这些方程，可以根据观察到的数据：输入序列和先前状态，去预测系统的未来状态



由于矩阵D类似于跳跃连接，因此在没有跳跃连接的情况下，SSM通常被视为如下



最终，我们可以通过下图统一这两个方程：

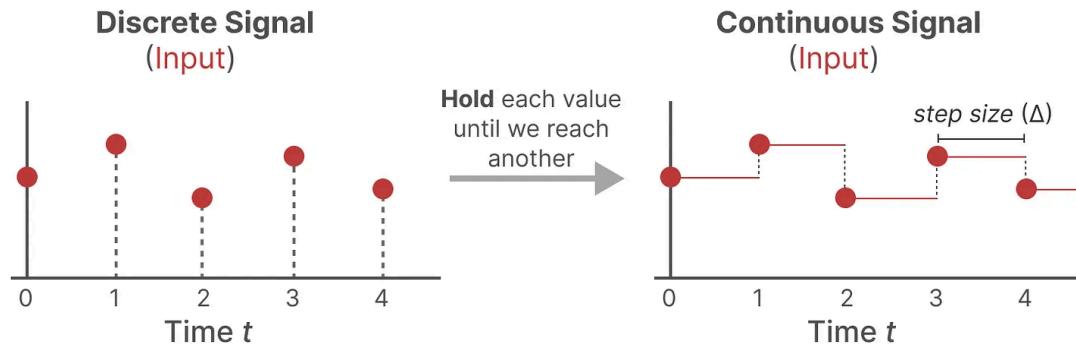


## 2.2 SSM to S4

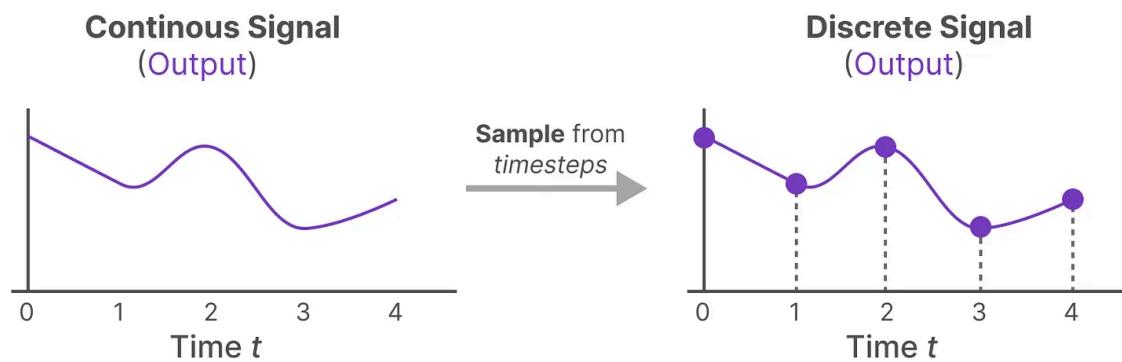
### 2.2.1 数据离散化

由于除了连续的输入之外，还会通常碰到离散的输入(如文本序列)，因此如果模型也能处理离散化数据则再好不过。

零阶保持技术(Zero-order hold technique)



- 首先，每次收到离散信号时，我们都会保留其值，直到收到新的离散信号，如此操作导致的结果就是创建了 SSM 可以使用的连续信号
- 保持该值的时间由一个新的可学习参数表示，称为步长(siz)—— $\Delta$  它代表输入的阶段性保持(resolution)
- 有了连续的输入信号后，便可以生成连续的输出，并且仅根据输入的时间步长对值进行采样



这些采样值就是我们的离散输出，且可以按如下方式做零阶保持

$$\text{Discretized Matrix A: } \bar{A} = e^{\Delta A}$$

$$\text{Discretized Matrix B: } \bar{B} = (\Delta A)^{-1}(e^{\Delta A} - I) * \Delta B$$

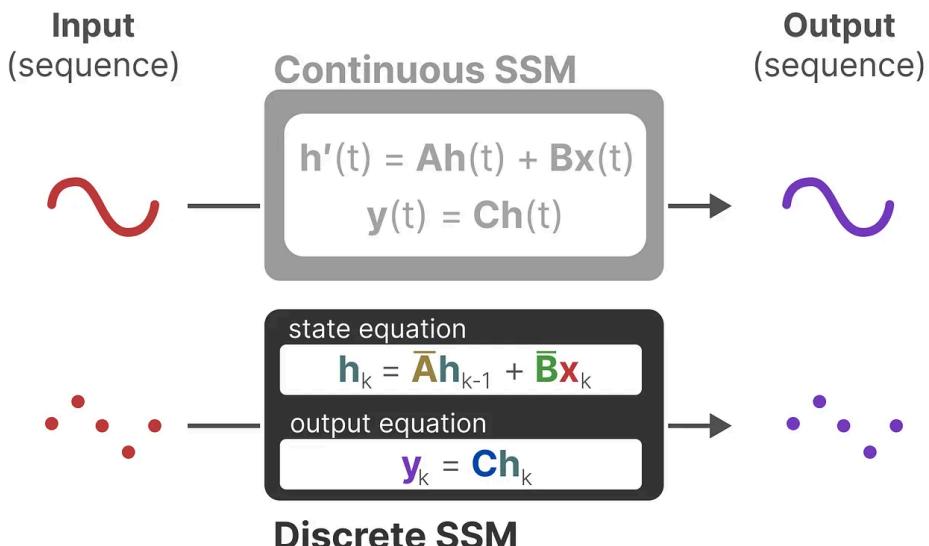
Discretized matrix  $A$

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

Discretized matrix  $B$

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

它们共同使我们能够从连续 SSM 转变为离散SSM，使得不再是函数到函数  $x(t) \rightarrow y(t)$ ，而是序列到序列  $x_k \rightarrow y_k$ ，所以你看到离散化的SSM时，不再带参数t.

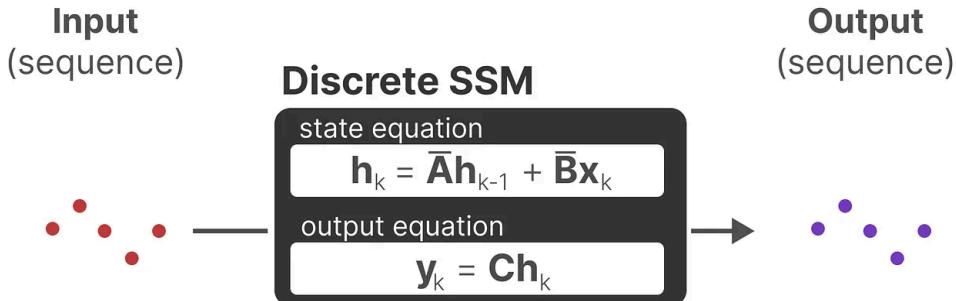


这里，矩阵和现在表示模型的离散参数(且这里使用  $k$ ，而不是  $t$  来表示离散时间步长)

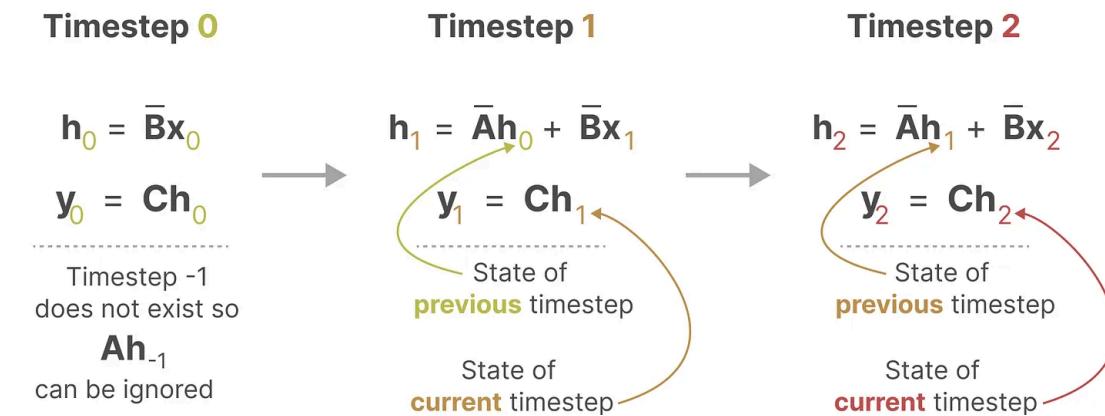
注意：我们在保存时，仍然保存矩阵的连续形式(而非离散化版本)，只是在训练过程中，连续表示被离散化

## 2.2.2 循环结构表示The Recurrent Representation

总之，离散 SSM 可以用离散时间步长重新表述问题

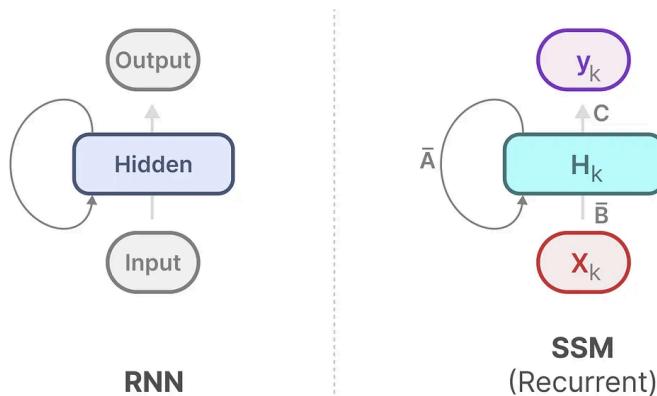


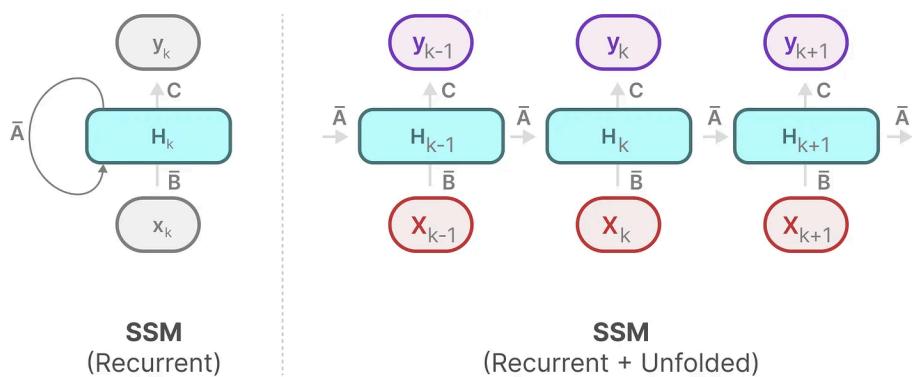
在每个时间步，都会涉及到隐藏状态的更新



$$\begin{aligned}
 \mathbf{y}_2 &= \mathbf{C}\mathbf{h}_2 \\
 &= \mathbf{C}(\bar{\mathbf{A}}\mathbf{h}_1 + \bar{\mathbf{B}}\mathbf{x}_2) \\
 &= \mathbf{C}(\bar{\mathbf{A}}(\bar{\mathbf{A}}\mathbf{h}_0 + \bar{\mathbf{B}}\mathbf{x}_1) + \bar{\mathbf{B}}\mathbf{x}_2) \\
 &= \mathbf{C}(\bar{\mathbf{A}}(\bar{\mathbf{A}} \cdot \bar{\mathbf{B}}\mathbf{x}_0 + \bar{\mathbf{B}}\mathbf{x}_1) + \bar{\mathbf{B}}\mathbf{x}_2) \\
 &= \mathbf{C}(\bar{\mathbf{A}} \cdot \bar{\mathbf{A}} \cdot \bar{\mathbf{B}}\mathbf{x}_0 + \bar{\mathbf{A}} \cdot \bar{\mathbf{B}}\mathbf{x}_1 + \bar{\mathbf{B}}\mathbf{x}_2) \\
 &= \mathbf{C} \cdot \bar{\mathbf{A}}^2 \cdot \bar{\mathbf{B}}\mathbf{x}_0 + \mathbf{C} \cdot \bar{\mathbf{A}} \cdot \bar{\mathbf{B}} \cdot \mathbf{x}_1 + \mathbf{C} \cdot \bar{\mathbf{B}}\mathbf{x}_2
 \end{aligned}$$

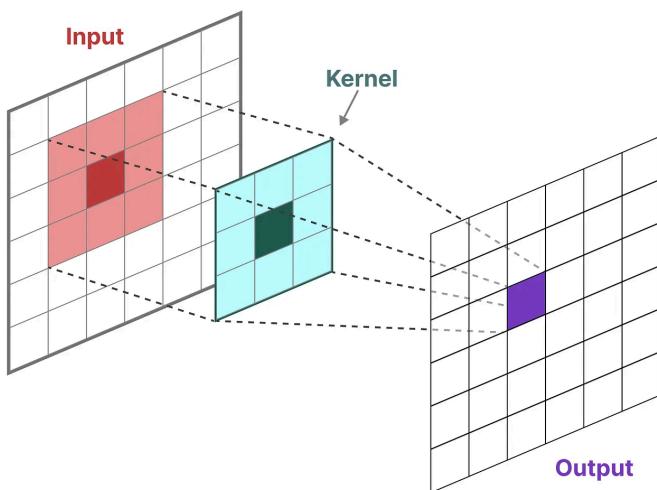
如此，便可以RNN的结构来处理



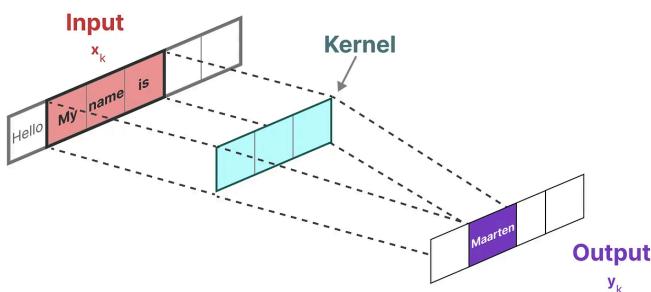


### 2.2.3 卷积结构表示The Convolution Representation

在经典的图像识别任务中，我们用过滤器(即卷积核kernels)来导出聚合特征，而SSM也可以表示成卷积的形式



由于我们处理的是文本而不是图像，因此我们需要一维视角



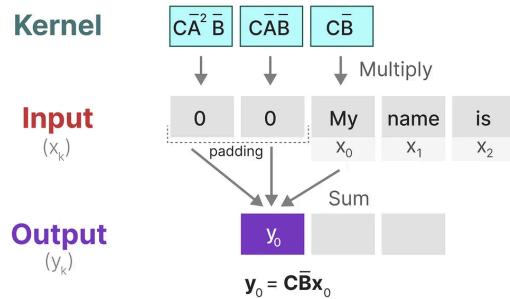
而用来表示这个“过滤器”的内核源自 SSM 公式

$$\text{kernel} \rightarrow \bar{\mathbf{K}} = (\bar{\mathbf{CB}}, \bar{\mathbf{CAB}}, \dots, \bar{\mathbf{CA}}^k \bar{\mathbf{B}}, \dots)$$

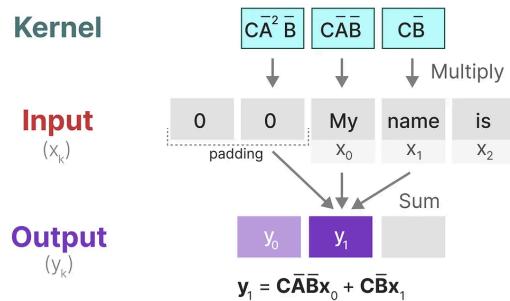
$$\mathbf{y} = \mathbf{x} * \bar{\mathbf{K}}$$

↑ output    ↑ input    ↑ kernel

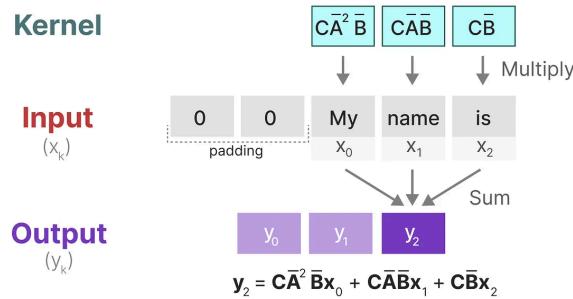
- 与卷积一样，我们可以使用 SSM 内核来检查每组token并计算输出



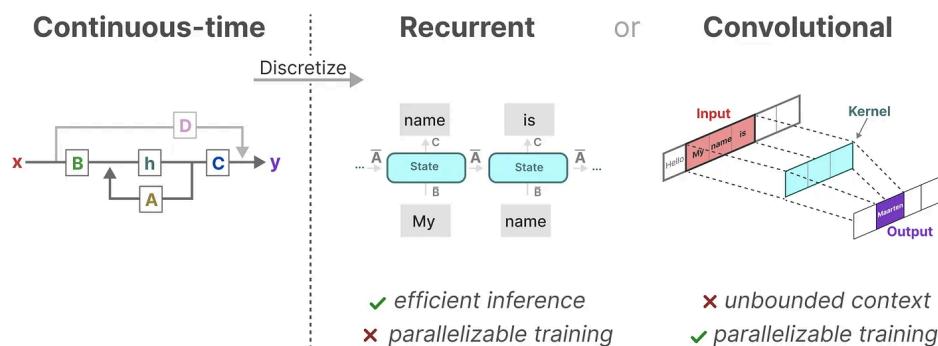
- 内核将移动一次以执行下一步的计算



- 最后一步，我们可以看到内核的完整效果

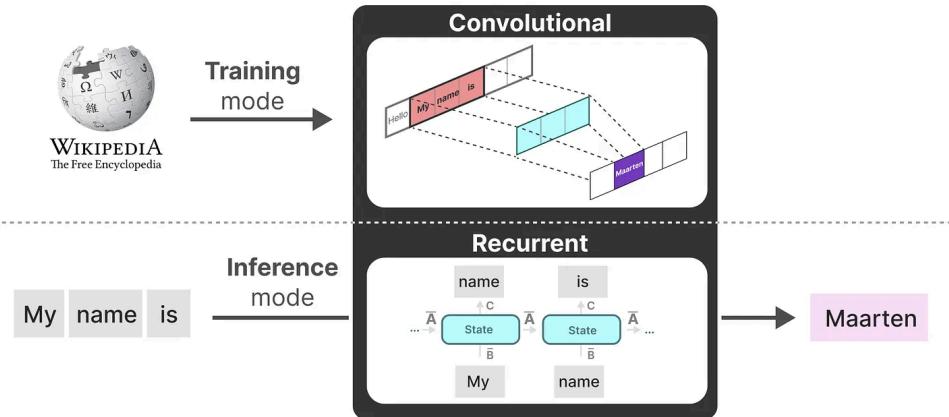


总结一下，将 SSM 表示为卷积的一个主要好处是它可以像卷积神经网络CNN一样进行并行训练。然而，由于内核大小固定，它们的推理不如 RNN 那样快速



**i** SSMs可以当做是RNN与CNN的结合，即**推理用RNN，训练用CNN**

## State Space Model

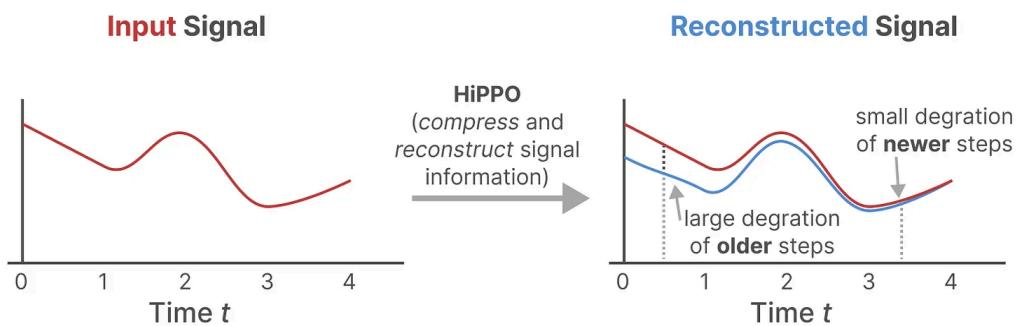


### 2.2.4 长距离依赖问题: HiPPO

如我们之前在循环表示中看到的那样，矩阵A捕获先前状态的信息来构建新状态。由于矩阵A只记住之前的几个token和捕获迄今为止看到的每个token之间的区别，特别是在循环表示的上下文中，因为它只回顾以前的状态。那么我们怎样才能以保留比较长的memory的方式创建矩阵A呢？

#### Hippo: High-order Polynomial Projection Operator

HiPPO尝试将当前看到的所有输入信号压缩为系数向量



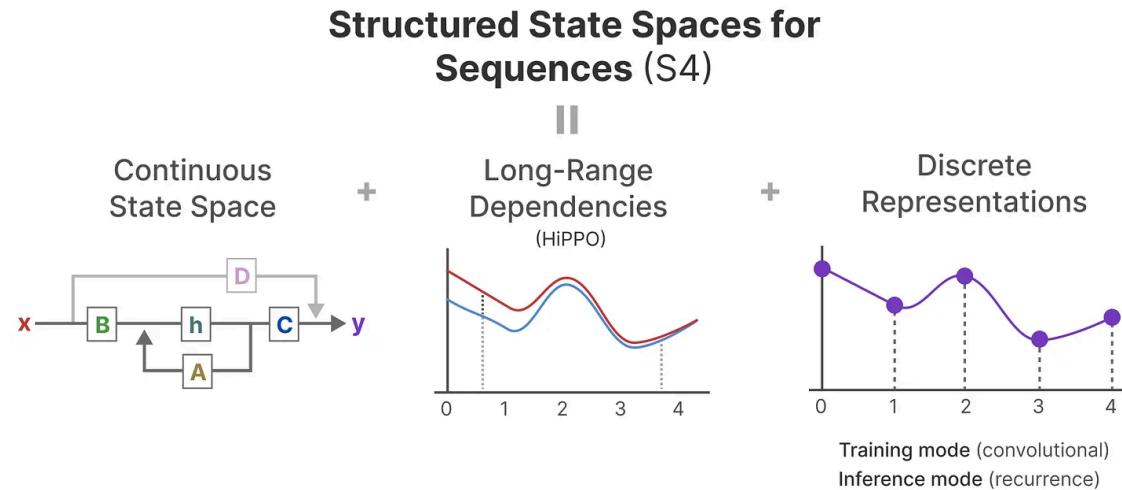
它使用矩阵构建一个“可以很好地捕获最近的token并衰减旧的token”状态表示，说白了，通过函数逼近产生状态矩阵  $A$  的最优解，其公式可以表示如下

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

HiPPO Matrix

1	0	0	0
1	2	0	0
1	3	3	0
1	3	5	4

如此，S4的定义就出来了：序列的结构化状态空间——**Structured State Space for Sequences**，一类可以有效处理长序列的SSM



### 3 Mamba: Linear-Time Sequence Modeling with Selective State Spaces

简言之，Mamba是一种状态空间模型(SSM)，建立在更现代的适用于深度学习的结构化SSM(简称S6)基础上，与经典架构RNN有相似之处。

#### 3.1 Mamba = Selection Mechanism + Hardware-aware Algorithm + SSM

与先前的研究相比，Mamba主要有三点创新：

- 对输入信息有选择性处理(Selection Mechanism)
- 硬件感知的算法(Hardware-aware Algorithm)：采用“并行扫描算法”而非“卷积”来进行模型的循环计算，但为了减少GPU内存层次结构中不同级别之间的IO访问，它没有具体化扩展状态。
- 更简单的架构：将SSM架构的设计与transformer的MLP块合并为一个块(combining the design of prior SSM architectures with the MLP block of Transformers into a single block)，来简化过去的深度序列模型架构，从而得到一个包含selective state space的架构设计

##### 3.1.1 From S4 to S6

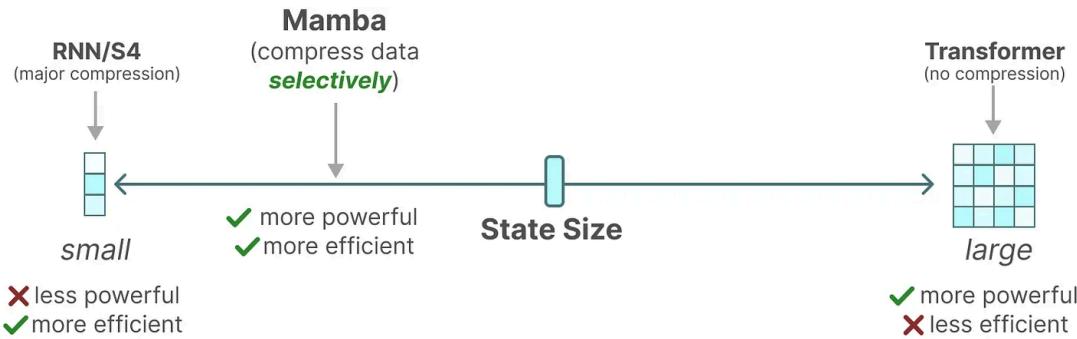
作者认为，序列建模的一个基础问题是把上下文压缩成更小的状态，从这个角度来看：

- transformer的注意力机制虽然有效果但效率不算很高，毕竟其需要显式地存储整个上下文(也就是KV缓存)，直接导致训练和推理消耗算力大
  - 好比，Transformer就像人类每写一个字之前，都把前面的所有字+输入都复习一遍，所以写的慢
- RNN的推理和训练效率高，但性能容易受到对上下文压缩程度的限制
  - 好比，RNN每次只参考前面固定的字数，写的快是快，但容易忘掉更前面的内容
- 而SSM的问题在于其中的矩阵A B C始终是不变的，无法针对不同的输入针对性的推理
- 最终，Mamba的解决办法是，相比SSM压缩所有历史记录，mamba设计了一个简单的选择机制，通过“参数化SSM的输入”，让模型对信息有选择性处理，以便关注或忽略特定的输入。这样一来，模型能够过滤掉与问题无关的信息，并且可以长期记住与问题相关的信息
  - 好比，Mamba每次参考前面所有内容的一个概括，越往后写对前面内容概括得越狠，丢掉细节、保留大意

总之，序列模型的效率与效果的权衡点在于它们对状态的压缩程度：

- 高效的模型必须有一个小的状态(比如RNN或S4)
- 而有效的模型必须有一个包含来自上下文的所有必要信息的状态(比如transformer)

而mamba为了兼顾效率和效果，选择性的关注必须关注的、过滤掉可以忽略的



## 回顾

在其前身S4中，其有4个参数( $\Delta, \mathbf{A}, \mathbf{B}, \mathbf{C}$ )

且它们都是固定的，不随输入变化(即与输入无关)，这些参数控制了以下两个阶段：

$$\begin{aligned} h'(t) &= \mathbf{A}h(t) + \mathbf{B}x(t) & (1a) \quad h_t &= \bar{\mathbf{A}}h_{t-1} + \bar{\mathbf{B}}x_t & (2a) \quad \bar{\mathbf{K}} &= (\mathbf{C}\bar{\mathbf{B}}, \mathbf{C}\bar{\mathbf{A}}\bar{\mathbf{B}}, \dots, \mathbf{C}\bar{\mathbf{A}}^k\bar{\mathbf{B}}, \dots) & (3a) \\ y(t) &= \mathbf{C}h(t) & (1b) \quad y_t &= \mathbf{C}h_t & (2b) \quad y &= x * \bar{\mathbf{K}} & (3b) \end{aligned}$$

- **第一阶段(1a 1b)**，通常采用固定公式 $\mathbf{A} = f\mathbf{A}(\Delta, \mathbf{A})$ 和 $\mathbf{B} = f\mathbf{B}(\Delta, \mathbf{A}, \mathbf{B})$ ，将“连续参数”( $\Delta, \mathbf{A}, \mathbf{B}$ )转化为“离散参数”( $\mathbf{A}, \mathbf{B}$ )，其中 $(f\mathbf{A}, f\mathbf{B})$ 称为离散化规则，且可以使用多种规则来实现这一转换。

例如下述方程中定义的零阶保持(ZOH)：

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

- **第二阶段(2a 2b， 和3a 3b)**，在参数由( $\Delta, \mathbf{A}, \mathbf{B}, \mathbf{C}$ )变换为( $\mathbf{A}, \mathbf{B}, \mathbf{C}$ )后，模型可以用两种方式计算，即线性递归(2)或全局卷积(3)
  - 模型通常使用卷积模式(3)可以进行高效的并行化训练，其中整个输入序列提前看到
  - 并切换到循环模式(2)以高效的自回归推理(其中输入每次只看到一个时间步)

## 关键：

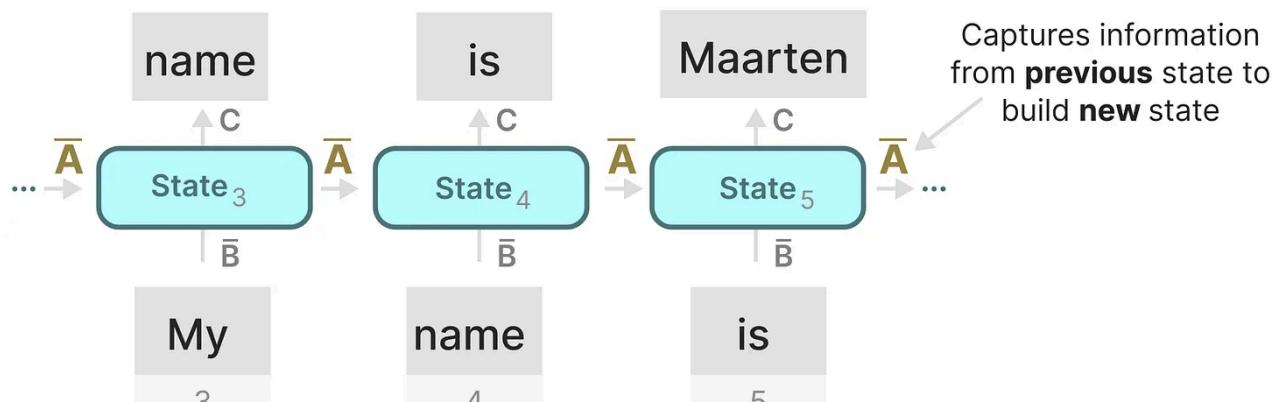
- $\Delta$  类似遗忘门

如mamba作者回复审稿人的一段话所说，“In general,  $\Delta$  controls the balance between how much to focus or ignore the current input. It is analogous to the role of the gate in Theorem 1, mechanically, a large  $\Delta$  resets the state and focuses on the current input, while a small  $\Delta$  persists(保持) the state and ignores the current input.”。说白了，较小的步长 $\Delta$ 会忽略特定单词，而更多地使用先前的上文，而较大的步长 $\Delta$ 会更多地关注输入单词而不是上文



- $\mathbf{B}$ 类似进RNN的memory
- $\mathbf{C}$ 类似出RNN的memory

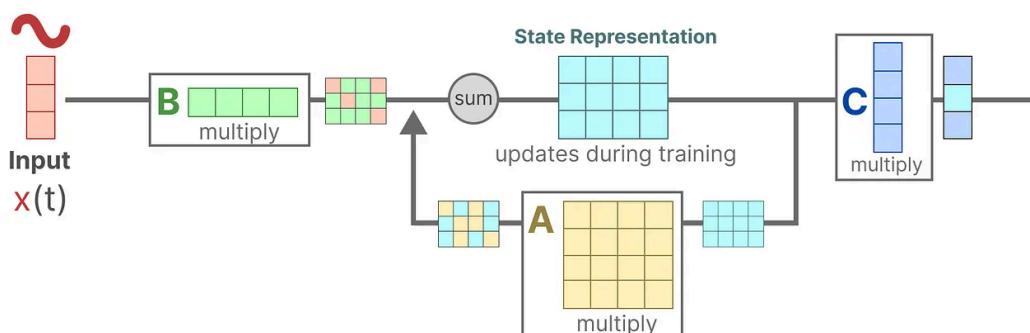
修改 $\mathbf{B}$ 和 $\mathbf{C}$ 可以允许模型更精细地控制是否让输入 $x$ 进入状态  $h$ ，或状态 $h$ 进入输出  $y$ ，所以  $\mathbf{B}$  和  $\mathbf{C}$  类似于 RNN 中的输入门和输出门



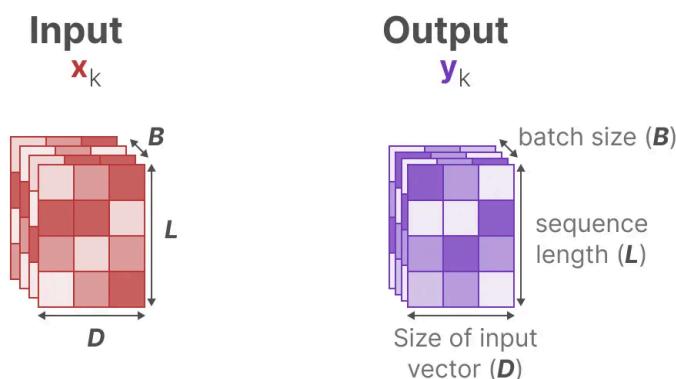
## SSM (Recurrent + Unfolded)

- $A$ 在每个hidden state维度上的作用可以不相同，起到multi-scale/fine-grained gating的作用，这也是LSTM网络里面用element-wise product的原因

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



为了对批量大小为 $B$ 、长度为 $L$ 、具有 $D$ 个通道(类似R G B三个通道)的输入序列  $x$  进行操作，SSM被独立地应用于每个通道：



在这种情况下，每个输入的总隐藏状态具有 $DN$ 维，在序列长度上计算它需要 $O(BLDN)$ 的时间和内存

最后，在Mamaba中，作者让  $B$  矩阵、  $C$  矩阵、  $\Delta$  成为输入的函数(即可学习或可训练的)，让模型能够根据输入内容自适应地调整其行为

**Algorithm 1 SSM (S4)**

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

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

**Algorithm 2 SSM + Selection (S6)**

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

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

在S4到S6的过程中：

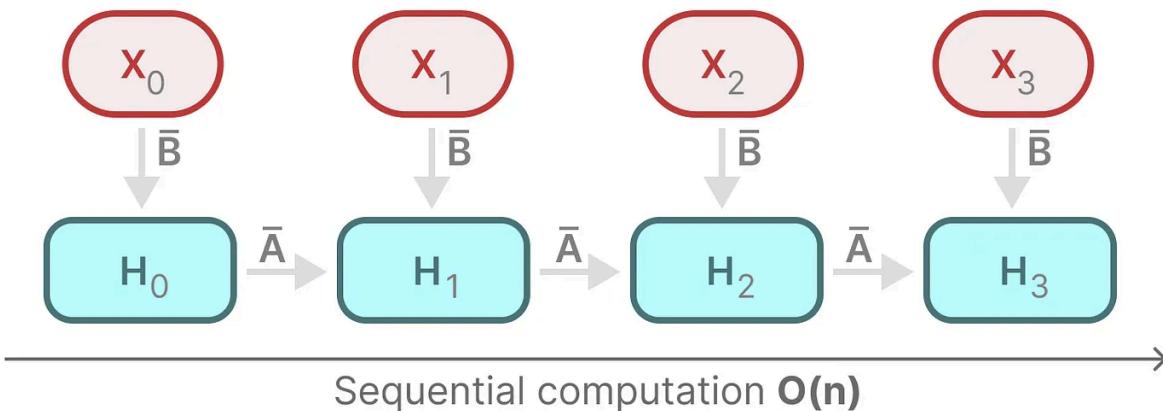
- 影响输入的  $B$  矩阵、影响状态的  $C$  矩阵的大小从原来的(D,N), 变成了(B,L,N) 「这三个参数分别对应batch size、sequence length、hidden state size」
- 且  $\Delta$  的大小由原来的D变成了(B,L,D)
- 且每个位置的  $B$  矩阵、  $C$  矩阵、  $\Delta$  都不相同, 这意味着对于每个输入token, 现在有不同的  $B$  矩阵、  $C$  矩阵、可以解决内容感知问题
- 进一步, 通过  $S_B(x) = \text{Linear}_N(x)$   $S_C(x) = \text{Linear}_N(x)$   $S_\Delta(x) = \text{Linear}_D(x)$   $T_\Delta = \text{softplus}$  来逐一将  $B$  矩阵、  $C$  矩阵、  $\Delta$  数据依赖化(data dependent) 「其中的Linear代表把D维的输入向量  $x$  经过一个线性层映射到 d 维」
- 虽然  $A$  没有变成data dependent, 但是通过SSM的离散化操作之后,  $(\bar{A}, \bar{B})$  会经过outer product变成(B, L, N, D)的data dependent张量, 算是以一种parameter efficient的方式来达到data dependent的目的
- $A$  离散化之后,  $\bar{A} = \exp(\Delta A)$  的数据依赖能够让整体的与输入相关

① 总之, Mamba通过合并输入的序列长度和批量大小来使矩阵B和C, 甚至步长 $\Delta$ 取决于输入(其意味着对于每个输入token, 现在有不同的B和C矩阵, 可以解决内容感知问题), 从而达到选择性地选择将哪些内容保留在隐藏状态以及忽略哪些内容的目标

### 3.1.2 并行扫描(parallel scan): 使得不用CNN也能并行训练

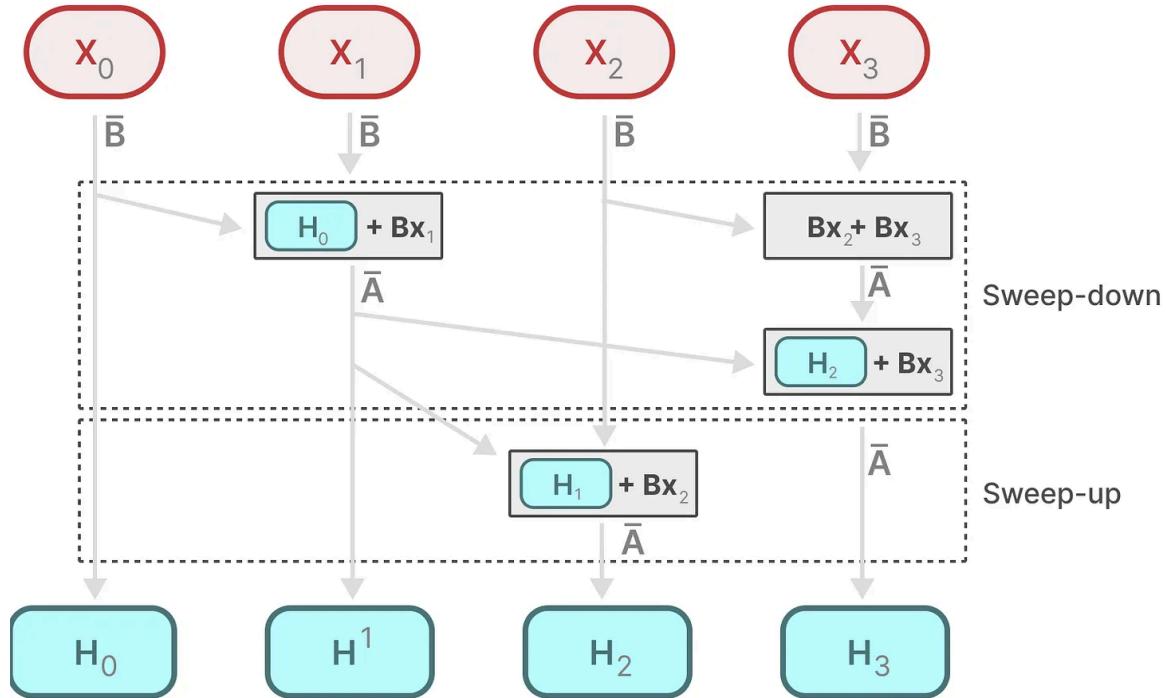
由于ABC这些矩阵现在是动态的了, 因此无法使用卷积表示来计算它们(CNN需要固定的内核), 因此, 我们只能使用循环表示, 如此也就而失去了卷积提供的并行训练能力

so, 为了实现并行化, 让我们探讨如何使用循环计算输出



每个状态, 比如  $H_1$  都是前一个状态比如  $H_0$  乘以  $\bar{A}$ , 加上当前输入  $X_1$  乘以  $\bar{B}$  的总和, 这就叫扫描操作(scan operation), 可以使用 for 循环轻松计算, 然这种状态之下想并行化是不可能的(因为只有在获取到前一个状态的情况下才能计算当前的每个状态)

mamba通过并行扫描(parallel scan)算法使得最终并行化成为可能, 其假设我们执行操作的顺序与关联属性无关



Parallel computation  $O(n/t)$

因此，我们可以分段计算序列并迭代地组合它们，即动态矩阵B和C以及并行扫描算法一起创建选择性扫描算法(selective scan algorithm)

为了方便大家更好的理解，我把相关推导再拆解一下，以更一目了然

- 首先， $H_1$ 和 $H_2$ 的计算很简单，如下所示

$$H_1 = \bar{A} \cdot H_0 + BX_1$$

$$H_2 = \bar{A} \cdot H_1 + BX_2$$

$$= \bar{A} \cdot (\bar{A}H_0 + BX_1) + BX_2$$

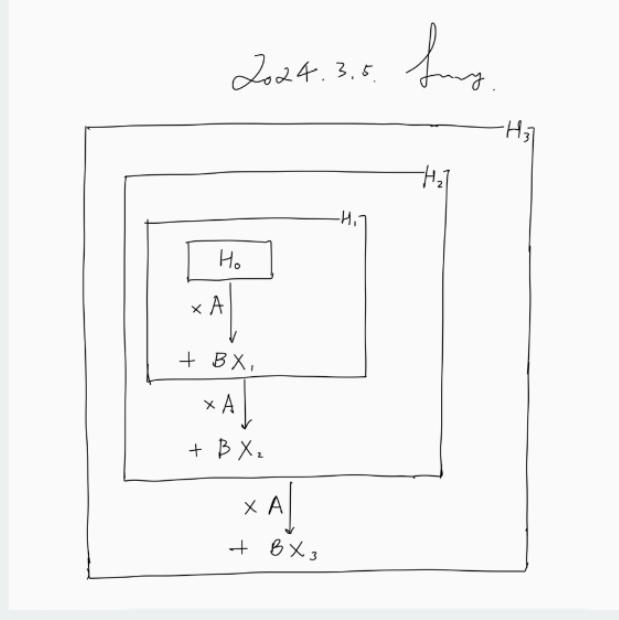
- 其次， $H_3$ 可以由 $H_2$ 直接计算得来，也可以由 $H_1$ 甚至 $H_0$ 计算得来

$$H_3 = \bar{A} \cdot H_2 + BX_3$$

$$= \bar{A} \cdot (\bar{A} \cdot H_1 + BX_2) + BX_3$$

$$= \bar{A} \cdot (\bar{A} \cdot (\bar{A}H_0 + BX_1) + BX_2) + BX_3$$

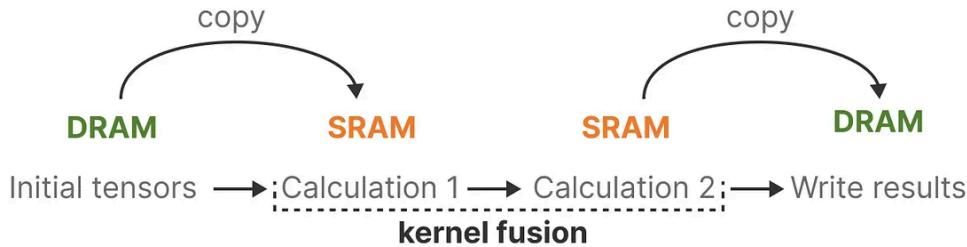
- 最后， $H_3$ 最终包含了之前 $H_1$ 、 $H_2$ 以及 $X_1$ 、 $X_2$ 、 $X_3$ 的信息，只是做了整体的压缩



### 3.1.3 硬件感知的状态扩展：借鉴Flash Attention

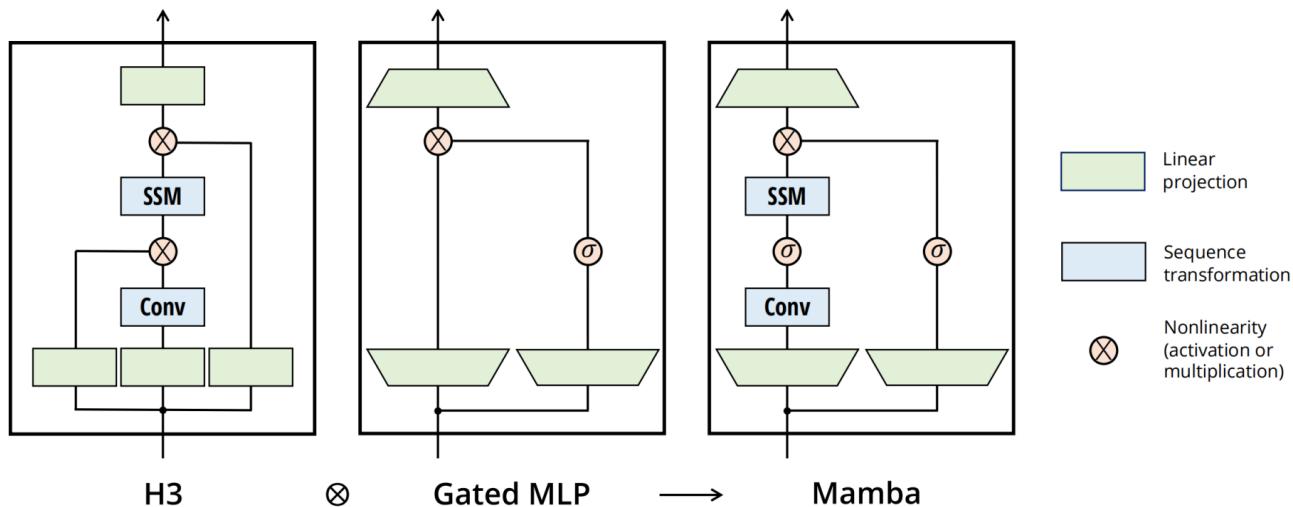
为了让传统的SSM在现代GPU上也能高效计算，Mamba中也使用了Flash Attention技术

- 利用内存的不同层级结构处理SSM的状态，减少高带宽但慢速的HBM内存反复读写这个瓶颈。就是限制需要从 DRAM 到 SRAM 的次数(通过内核融合kernel fusion来实现)，避免一有个结果便从SRAM写入到DRAM，而是待SRAM中有一批结果再集中写入DRAM中，从而降低来回读写的次数



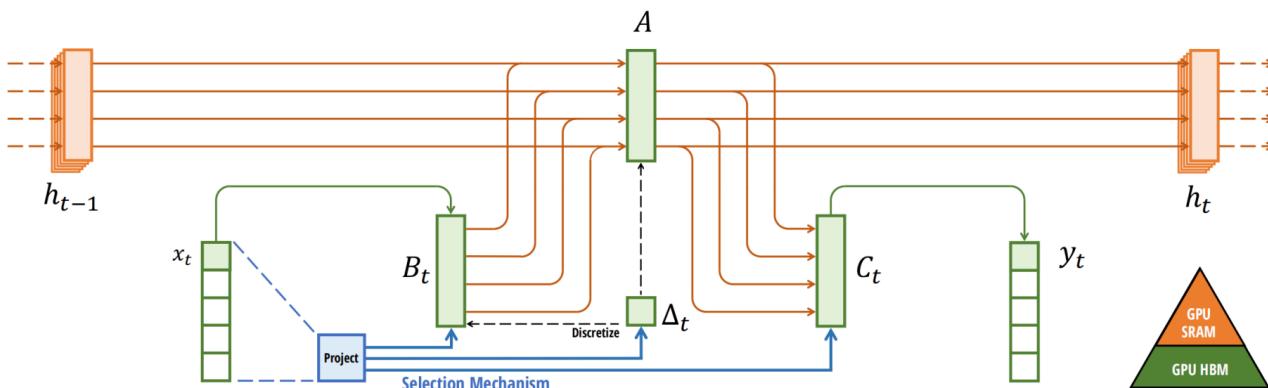
### 3.1.4 简化的SSM架构及最终的整体流程

将大多数SSM架构比如H3的基础块，与现代神经网络比如transformer中普遍存在的门控MLP相结合，组成新的Mamba块，重复这个块，与归一化和残差连接结合，便构成了Mamba架构



最终在更高速的SRAM内存中执行离散化和递归操作，再将输出写回HBM，具体来说

**Selective State Space Model  
with Hardware-aware State Expansion**

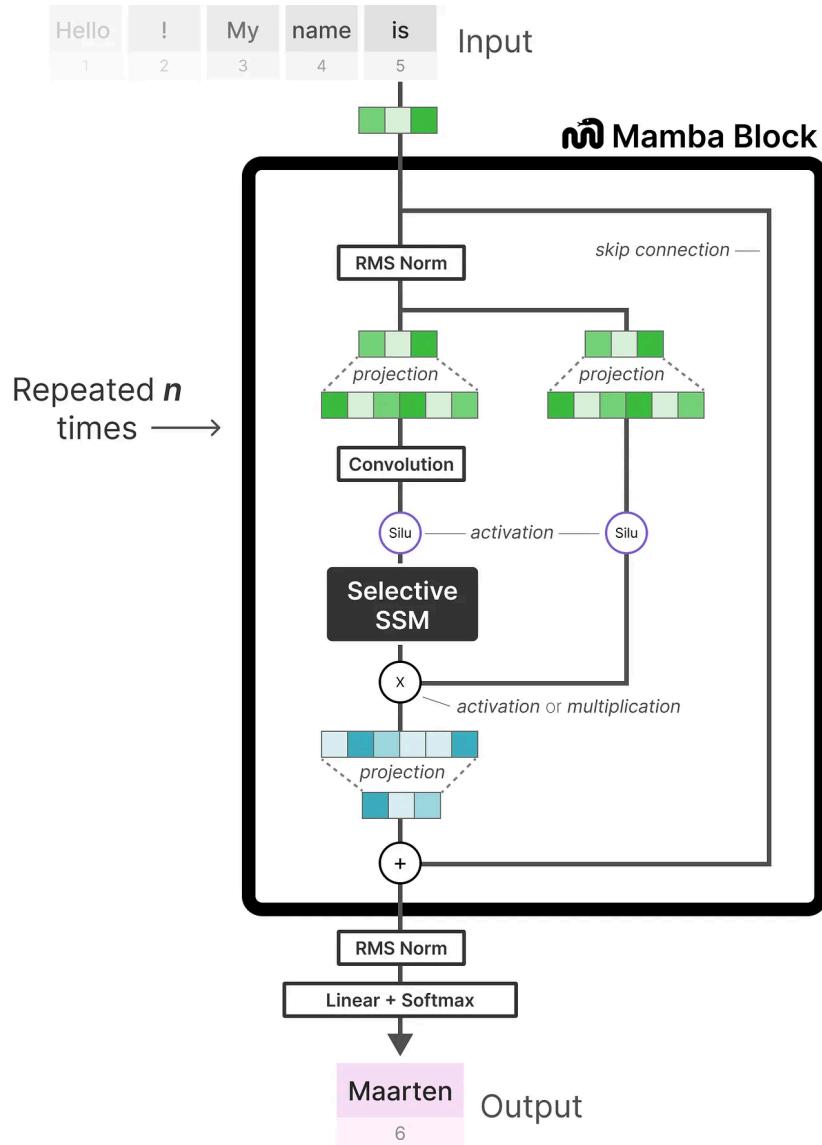


- 不是将大小为(B,L,D,N)的扫描输入进GPU HBM(高带宽内存)中，而是直接将SSM参数从慢速HBM加载到快速SRAM中
- 然后，在SRAM中进行离散化，得到(B,L,D,N)的A,B
- 接着，在SRAM中进行scan(通过上一节4.1.2节介绍的并行扫描算法实现并行化)，得到(B,L,D,N)的输出
- 最后，multiply and sum with C，得到(B,L,D)的最终输出写回HBM

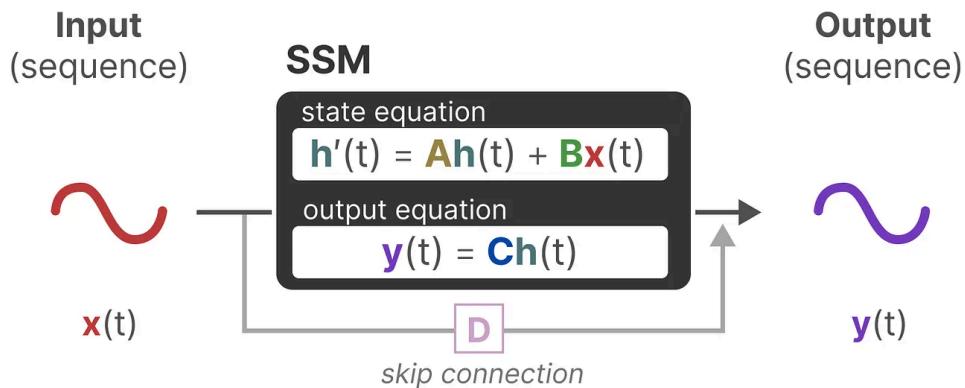
## 3.2 mamba的应用实例与一般性的实验结果

### 3.2.1 通过mamba预测下一个token的示例

首先进行线性投影以扩展输入嵌入，然后，在应用选择性 SSM之前先进行卷积(以防止独立的token计算)



其中的“选择性SSM(即Selective SSM)”具有以下属性:



Recurrent SSM通过离散化创建循环SSM

HiPPO对矩阵A进行初始化A以捕获长程依赖性

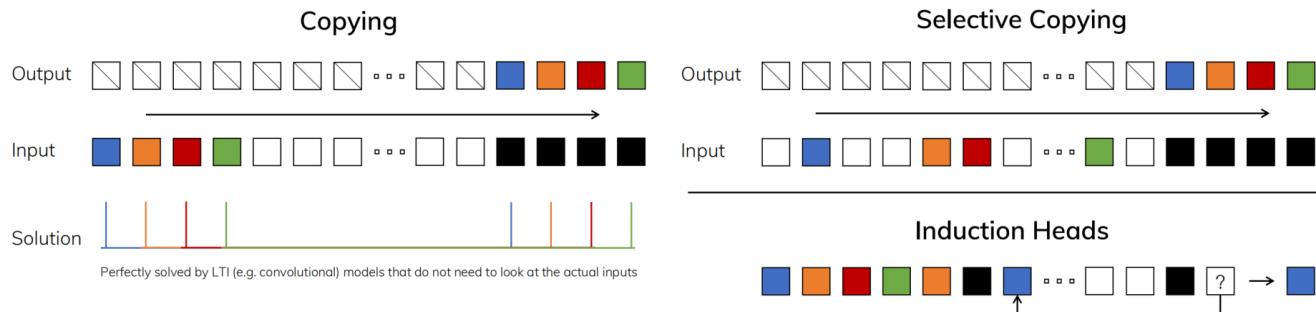
选择性扫描算法(Selective scan algorithm)选择性压缩信息

硬件感知算法(Hardware-aware algorithm)加速计算

最后，包含归一化层和用于选择“预测的token”的softmax

### 3.2.2 三个任务的对比：copying、selective copying、induction heads

如下图所示，有三个任务



(左)复制任务的标准版本涉及输入和输出元素之间的固定间距，可以通过线性递归和全局卷积等时不变模型轻松解决

(Left) The standard version of the Copying task involves constant spacing between input and output elements and is easily solved by time-invariant models such as linear recurrences and global convolutions.

(右上)选择性复制任务在输入之间具有随机间距，需要使用时变模型，在内容上能够灵活地选择记忆或忽略输入

(Right Top) The Selective Copying task has random spacing in between inputs and requires time-varying models that can selectively remember or ignore inputs depending on their content.

相当于选择性复制任务通过改变“要记忆的tokens的位置”来改进纯粹的复制任务。它需要内容感知推理，以便能够记住相关的标记(有色)，并过滤掉不相关的标记(白色)

The Selective Copying task modifies the popular Copying task (Arjovsky, Shah, and Bengio 2016) by varying the position of the tokens to memorize. It requires content-aware reasoning to be able to memorize the relevant

(右下)归纳头部任务是联想回忆的一个例子，需要根据上下文检索答案，这是LLM关键的能力

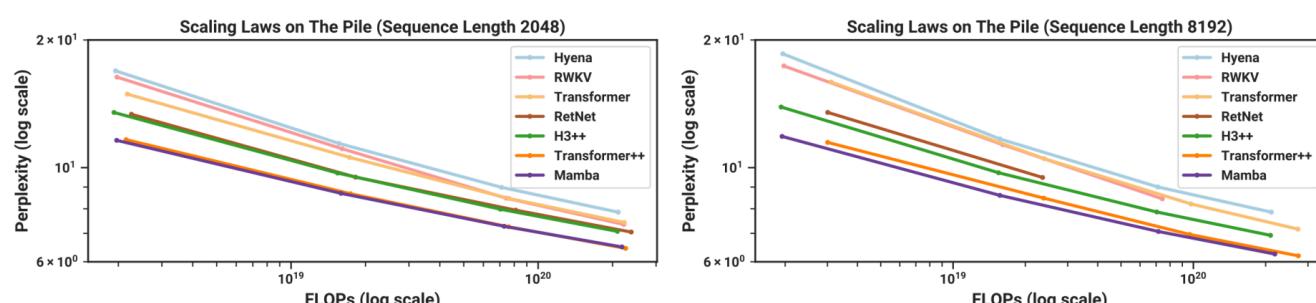
(Right Bottom) The Induction Heads task is an example of associative recall that requires retrieving an answer based on context, a key ability for LLMs.

其实，归纳头部任务是一种众所周知的机制，据推测可以解释LLMs的大部分上下文学习能力(Olsson et al. 2022)。它需要上下文感知的推理，以便知道何时在适当的上下文中产生正确的输出(黑色)

The Induction Heads task is a well-known mechanism hypothesized to explain the majority of in-context learning abilities of LLMs (Olsson et al. 2022). It requires context-aware reasoning to know when to produce the correct output in the appropriate context (black)

### 3.2.3 实验结果

Mamba在Chinchilla缩放定律下预训练时，语言任务优于同类开源模型



下游任务上，每个规模尺寸的Mamba都是同类最佳，并且通常与两倍规模的基线性能匹配，特别是当序列长度增加到512k时，相比使用FlashAttention-2的Transformer快几个数量级，而且不会内存不足

