Mamba: Zero to Hero

Contents:

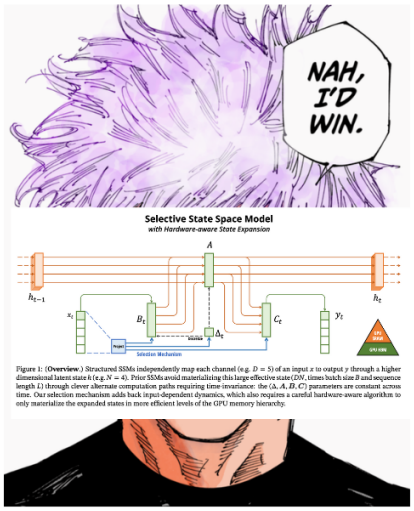
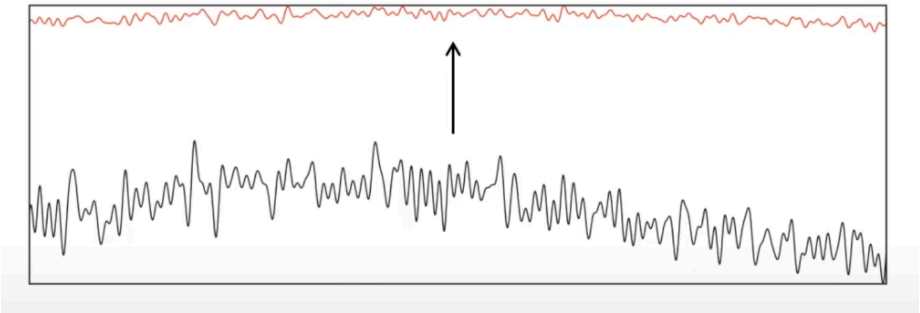
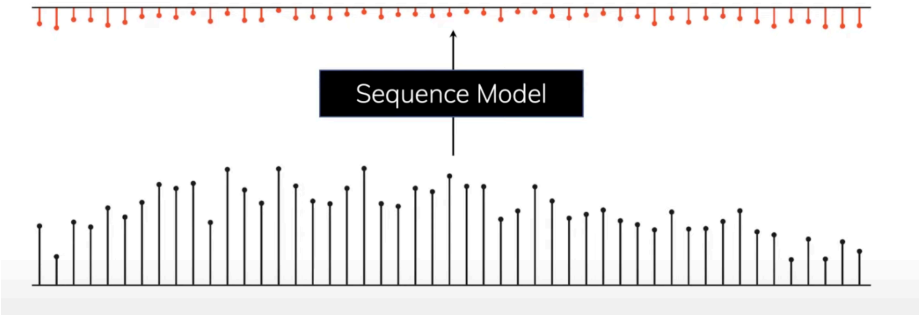


Figure 1: (**Overview.**) Structured SSMs independently map each channel (e.g. $D = 5$) of an input x to output y through a higher dimensional latent state h (e.g. $N = 4$). Prior SSMs avoid materializing this large effective state (DN, times batch size B and sequence length L) through clever alternate computation paths requiring time-invariance: the (Δ, A, B, C) parameters are constant across time. Our selection mechanism adds back input-dependent dynamics, which also requires a careful hardware-aware algorithm to only materialize the expanded states in more efficient levels of the GPU memory hierarchy.

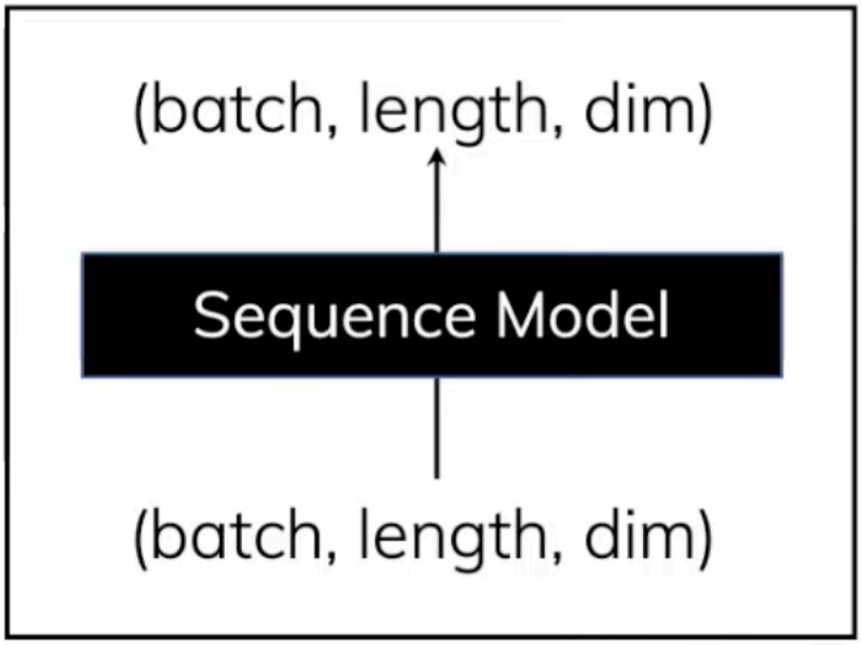
Throughout RNN and Attention
I alone am the Sub-Quadratic One

1.0. Sequence Modelling

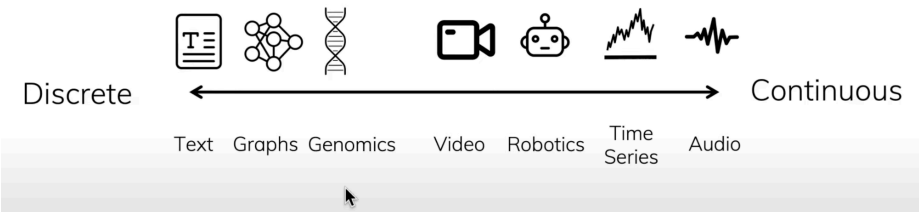




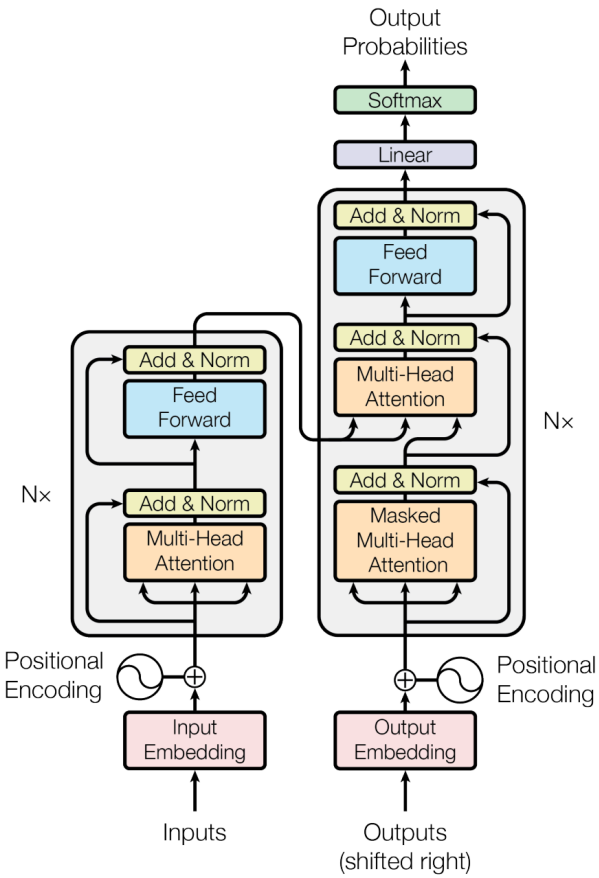
- Mapping an **input sequence** to the **output sequence**



1.1 What kind of data is used?



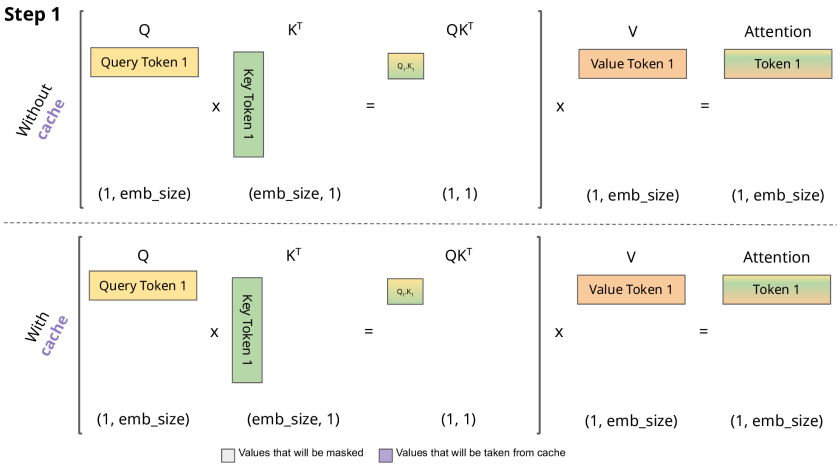
1.2 Transformers



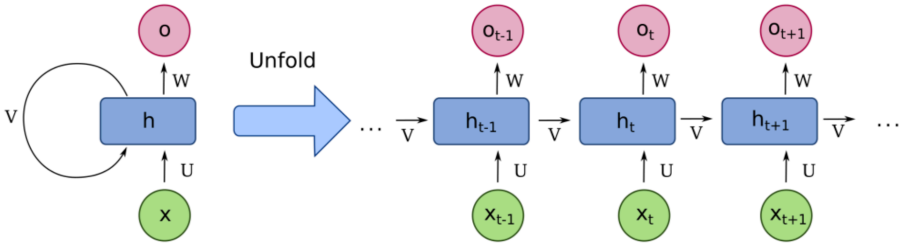
$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

- Limited context length

- Training: $O(N^2)$ + Parallizable architecture
- Generation: $O(N^2)$ but with KV Cache it gets to $O(N)$ + extra memory head / per token



1.3. RNN/LSTM



$$h_t = g_1(\bar{A}h_{t-1} + \bar{B}x_t)$$

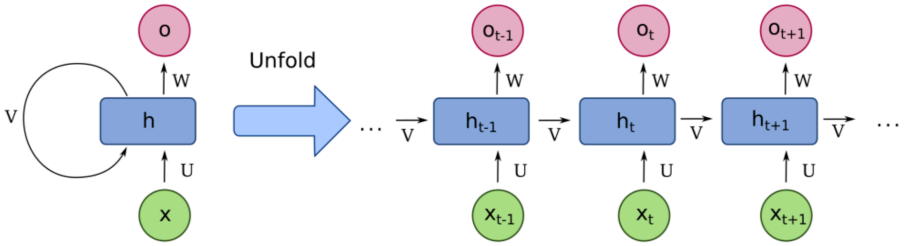
$$y_t = g_2(Ch_t)$$

- Theoretically: Infinite Context length
- Training: $O(N)$ + Non-parallizable architecture
- Generation: $O(1)$ (constant) for each token
- Vanishing/exploding gradient

1.4 Overall Issues

- Limited Context Window
- Non-parallizable model training and quadratic/sub quadratic Inference time
- Low performance architectures (RNN/LSTM)

2.0. Linear RNN



$$h_t = \bar{A}h_{t-1} + \bar{B}x_t$$

$$y_t = Ch_t$$

2.1 Training

At T = 0:

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

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

At T = 1:

$$h_1 = \bar{A}h_0 + \bar{B}x_1 = \bar{A}\bar{B}x_0 + \bar{B}x_1$$

$$y_1 = Ch_1 = C\bar{A}\bar{B}x_0 + C\bar{B}x_1$$

At T = 2:

$$h_2 = \bar{A}h_1 + \bar{B}x_2 = \bar{A}^2\bar{B}x_0 + \bar{A}\bar{B}x_1 + \bar{B}x_2$$

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

Generalizing the above pattern:

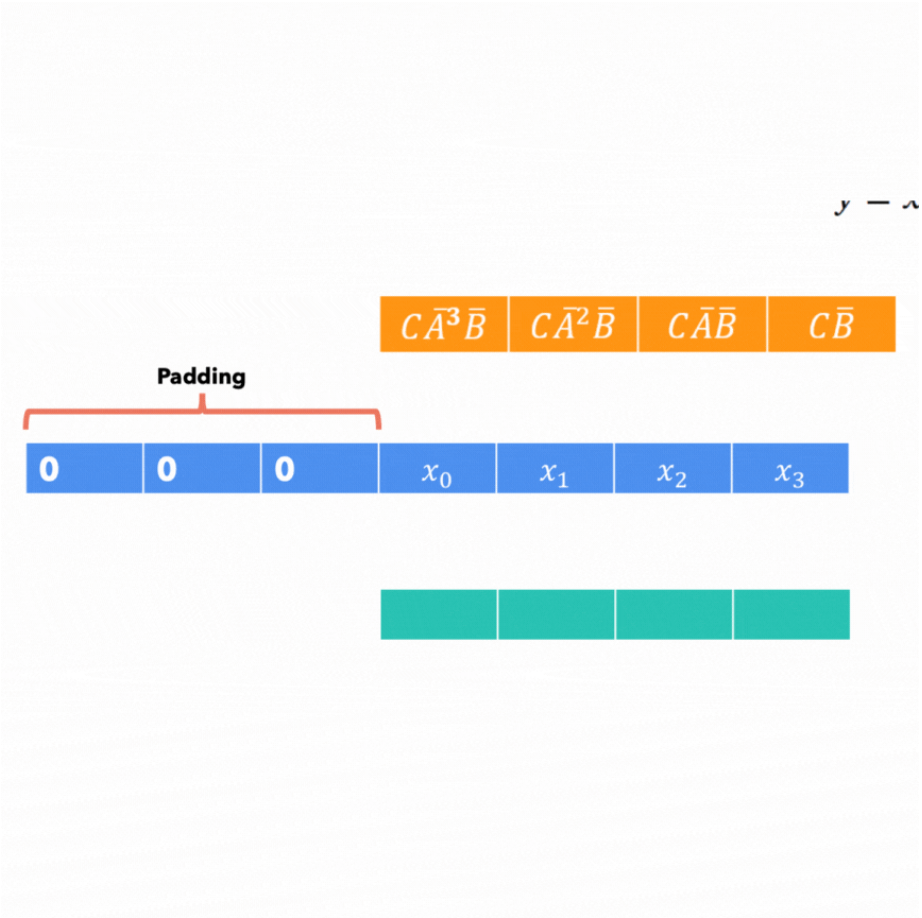
$$y_k = C\bar{A}^k\bar{B}x_0 + C\bar{A}^{k-1}\bar{B}x_1 + ... + C\bar{A}\bar{B}x_{k-1} + C\bar{B}x_k$$

2.1.1. Convolution Operation

This is interesting, given the formula above it can be computed using convolution operation over input $x(t)$. We can define the 1D kernel $\bar{K}$ and output y as:

$$\text{Kernel } (\bar{K}) = (C\bar{B}, C\bar{A}\bar{B}, ..., C\bar{A}^k\bar{B})$$

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



This 1D convolution operation is parallliziable, since the value of $\bar{K}$ is fixed, it can be easily pre-computed.

- Value of $\bar{K}$ is fixed because RNN uses the same parameter matrix for all tokens
- The linear kernel convolution on data is O(N) but doing it in Fourier space it is just multiplication O(NlogN) + internal accelerators optimization

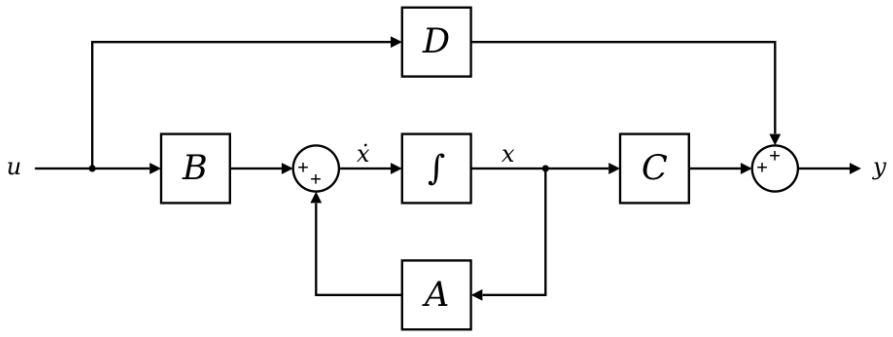
2.2. Summary

- Training Speed: Fast (Paralleliazable Convolution)
- Generation Speed: Fast (Constent per token, due to recurrence inference)
- Accuracy: Very Poor (Barely learns)

3.0. State Space Models (SSM, S4)

3.1. SSMs and where they came from?

General:



- Concept of SSMs are taken from the "Control System" field, The core idea of state-space systems is to represent a system's *dynamics*. This means capturing how the system changes over time.
- Many physical systems we model (electrical circuits, mechanical systems, chemical processes) operate in continuous time.
-
- State Space Equations are given as:
-

$$\frac{dx}{dt} = Ax(t) + Bu(t)$$

$$y(t) = Cx(t) + Du(t)$$

A, B, C and **D** are the learnable parameters, **D** can be ignored here (consider as 0) since from the architecture point of view it is nothing but a skip-connection

Sequence Modelling

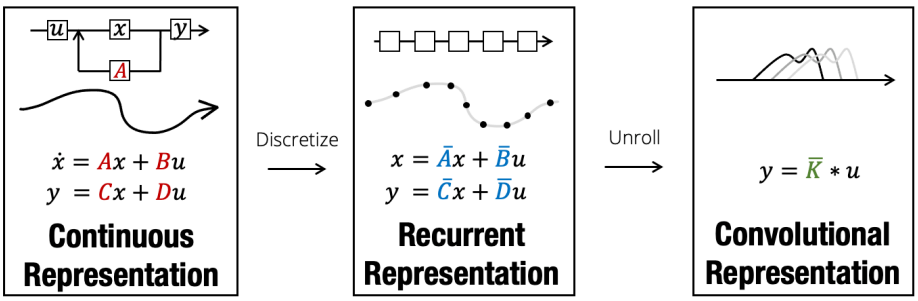
- SSMs are a continuous variant of Linear RNN block
- They model the state of tokens in sequence

- They are Linear and can be Time Invariant (S4) or Time Variant (S6/mamba)
- Desgined specially for handling long context
-

$$\frac{dh}{dt} = Ah(t) + Bx(t)$$

$$y(t) = Ch(t)$$

3.2. Fundamentals of Discritization



- We deal with descrete data, so need to map continuous state space to descrete.
- Looking on the equation only the parameter **A** and **B** influence the state **h**, so they need to be converted.

> Discretization has deep connections to continuous-time systems which can endow them with additional properties such as resolution invariance (Nguyen, Goel, et al. 2022) and automatically ensuring that the model is properly normalized (Gu, Johnson, Timalsina, et al. 2023; Orvieto et al. 2023). It also has connections to gating mechanisms of RNNs (Gu, Gulcehre, et al. 2020; Tallec and Ollivier 2018) which we will revisit in Section 3.5. However, from a mechanical point of view discretization can simply be viewed as the first step of the computation graph in the forward pass of an SSM. Alternate flavors of SSMs can bypass the discretization step and parameterize $(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}})$ directly instead (Zhang et al. 2023), which may be easier to reason about.

-

On the very fundamental level, we can write:

$$\lim_{\Delta \to 0} \frac{h(t+\Delta) - h(t)}{\Delta} = \frac{dh}{dt}$$

We can approximate the value of $h(t+\Delta)$ by choosing a small time step Δ as:

$$h(t+\Delta) \cong h'(t)\Delta + h(t)$$

From the State space equation:

$$h(t+\Delta) = h(t)(I + \Delta A) + \Delta Bx(t)$$

$$h(t+\Delta) = \bar{A}h(t) + \bar{B}x(t)$$

In the paper, authors used **Zero-Order Hold** (ZOH) rule to discretize the system, but the intent is same

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).

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

Now our system has 4 Parameters (Δ, A, B, C) , lets see what each of them represents.

$$h_t = \bar{A}h_{t-1} + \bar{B}x_t$$

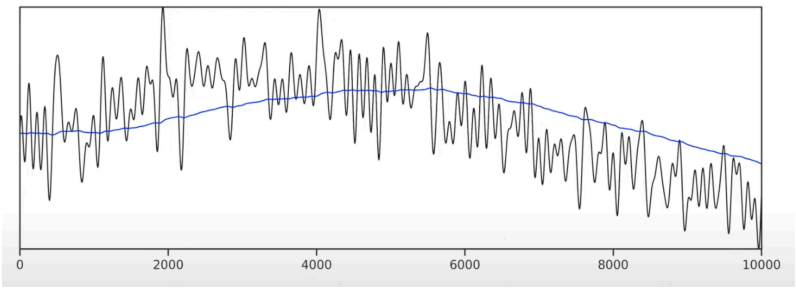
$$y_t = Ch_t$$

- Δ Represents the time step or the discretization interval in discrete-time models.
- **A**(State Transition Matrix): Describes how the current state of the system influences its next state.
- **B**(Control/Input Matrix): Maps how external inputs or control actions affect the state of the system.
- **C**(Output Matrix): Relates the internal state of the system to the outputs or observations that can be measured.

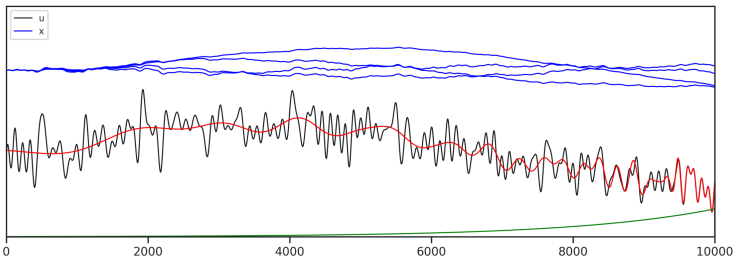
3.3. Importance of Matrix A

- Recalling the Exponential Moving Average as:
  –

$$h_t = (1 - \alpha)h_{t-1} + \alpha x_t$$



- Context representation for sequence modelling



–

2.2 Addressing Long-Range Dependencies with HiPPO

Prior work found that the basic SSM (1) actually performs very poorly in practice. Intuitively, one explanation is that linear first-order ODEs solve to an exponential function, and thus may suffer from gradients scaling exponentially in the sequence length (i.e., the vanishing/exploding gradients problem [32]). To address this problem, the LSSL leveraged the HiPPO theory of continuous-time memorization [16]. HiPPO specifies a class of certain matrices $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ that when incorporated into (1), allows the state $x(t)$ to memorize the history of the input $u(t)$. The most important matrix in this class is defined by equation (2), which we will call the HiPPO matrix. For example, the LSSL found that simply modifying an SSM from a random matrix $\boldsymbol{A}$ to equation (2) improved its performance on the sequential MNIST benchmark from 60% to 98%.

$$\text{(HiPPO Matrix)} \qquad \boldsymbol{A}_{nk} = -\begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases} \qquad (2)$$

–

– Basically gives a lower triangluar matrix structure (**initialization based on the HiPPO matrix**)

–

$$\mathbf{A} = \begin{bmatrix} 1 & & & & & & & & \\ -1 & 2 & & & & & & & \\ 1 & -3 & 3 & & & & & & \\ -1 & 3 & -5 & 4 & & & & & \\ 1 & -3 & 5 & -7 & 5 & & & & \\ -1 & 3 & -5 & 7 & -9 & 6 & & & \\ 1 & -3 & 5 & -7 & 9 & -11 & 7 & & \\ -1 & 3 & -5 & 7 & -9 & 11 & -13 & 8 & \\ \vdots & & & & & & & & \ddots \end{bmatrix}$$

– This matrix is not normal, but it can be decomposed as a normal matrix plus a matrix of lower rank.

3.4. Convoluition operation

We need to notice that our parameters (Δ, A, B, C) are time invariant $(\Delta, A, B, C) \to (\bar{A}, \bar{B}, C)$

$$h_t = \bar{A} h_{t-1} + \bar{B} x_t$$

$$y_t = C h_t$$

As per the computation, we can use the same convolution trick (discussed in Linear RNN) to effectively compute all the hidden states per token of an input sequence

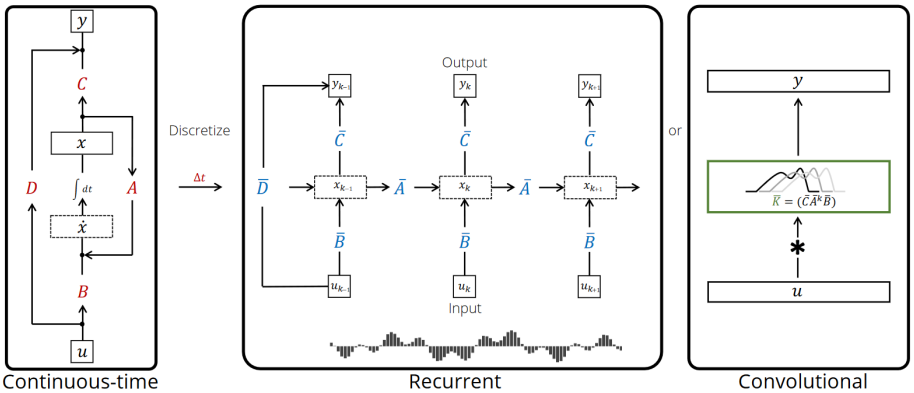
$$\text{Kernel } (\bar{K}) = (C\bar{B}, C\bar{A}\bar{B}, ..., C\bar{A}^k\bar{B})$$

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

Algorithm 1 SSM (S4)

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

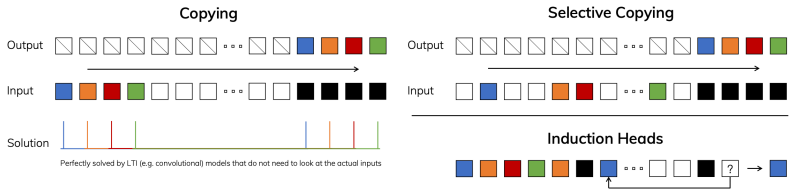
1: $\boldsymbol{A} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
3: $\boldsymbol{C} : (\mathtt{D}, \mathtt{N}) \leftarrow$ Parameter
4: $\Delta : (\mathtt{D}) \leftarrow \tau_\Delta$(Parameter)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



4.0. Mamba (Improving S4 to S6)

4.1. The Objective

- SSMs lacks the performance on Sequence modelling on dense modalities (language and genomics)
- Time Invariant parameters does not allow information selection
- SSM lack content-aware reasoning
- Attention explicitly does not compress context at all.
- Synthetic Tasks:
  - 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 tokens (colored) and filter out the irrelevant ones (white).
  - 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).



4.2. Selective SSM (S6)

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

We specifically choose $s_B(x) = \text{Linear}_N(x)$, $s_C(x) = \text{Linear}_N(x)$, $s_\Delta(x) = \text{Broadcast}_D(\text{Linear}_1(x))$, and $\tau_\Delta = \text{softplus}$, where Linear_d is a parameterized projection to dimension d. The choice of s_Δ and τ_Δ is due to a connection to RNN gating mechanisms explained in Section 3.5.

Variable Spacing. Selectivity allows filtering out irrelevant noise tokens that may occur between inputs of interest. This is exemplified by the Selective Copying task, but occurs ubiquitously in common data modalities, particularly for discrete data – for example the presence of language fillers such as "um". This property arises because the model can mechanistically filter out any particular input x_t, for example in the gated RNN case (Theorem 1) when $g_t \to 0$.

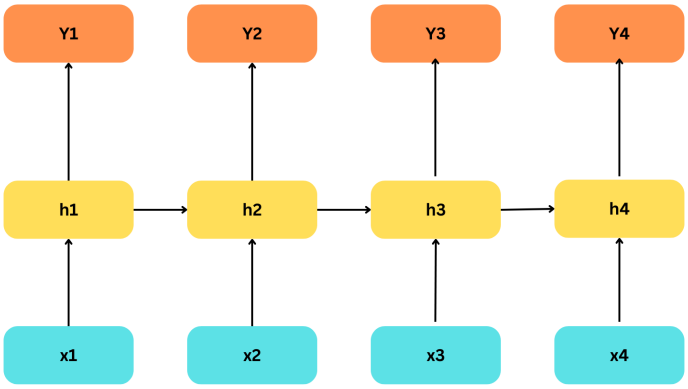
Filtering Context. It has been empirically observed that many sequence models do not improve with longer context (F. Shi et al. 2023), despite the principle that more context should lead to strictly better performance. An explanation is that many sequence models cannot effectively ignore irrelevant context when necessary; an intuitive example are global convolutions (and general LTI models). On the other hand, selective models can simply reset their state at any time to remove extraneous history, and thus their performance in principle improves monotonicly with context length (e.g. Section 4.3.2).

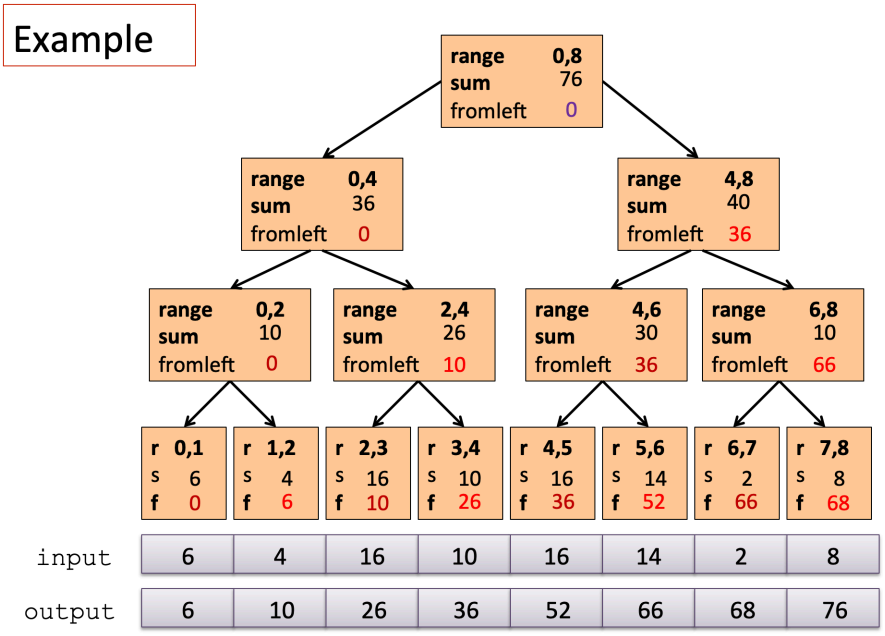
Boundary Resetting. In settings where multiple independent sequences are stitched together, Transformers can keep them separate by instantiating a particular attention mask, while LTI models will bleed information between the sequences. Selective SSMs can also reset their state at boundaries (e.g. $\Delta_t \to \infty$ or Theorem 1 when $g_t \to 1$). These settings may occur artificially (e.g. packing documents together to improve hardware utilization) or naturally (e.g. episode boundaries in reinforcement learning (Lu et al. 2023)).

Additionally, we elaborate on effects of each selective parameter.

- **Selective Nature** is only allowed to (Δ, B, C)
- **Selective Interpretation of** (Δ, A, B, C) **parameters:**
  - Δ: It controls the balance between how much to **focus** or **ignore** the current input x_t. SSMs can be interpreted as a continuous system discretized by a timestep Δ, and in this context the intuition is that large $\Delta \to \infty$ represents the system focusing on the current input for longer (thus "selecting" it and forgetting its current state) while a small $\Delta \to 0$ represents a transient input that is ignored.
  - **A**: The interaction of A to the model is via Δ as $\bar{A} = exp(\Delta A)$ (Zero Hold Discretization), so selectivity in Δ is enough to ensure the selectivity of A
  - **B and C**: Selectivity is to filter out the irrelevant information out of context to ensure the efficient compressed state. Making B and C selective allows finer-grained control over whether to let an input x_t into the state h_t or the state into the output y_t
- **Issue: Parallized Convolution for computation won't work (SSM/Linear RNN)**

4.2.1 Parallel Recurrence Scan (Prefix Sum Variant)





4.2.2 Hardware Aware State Expansion & Kernel Fusion

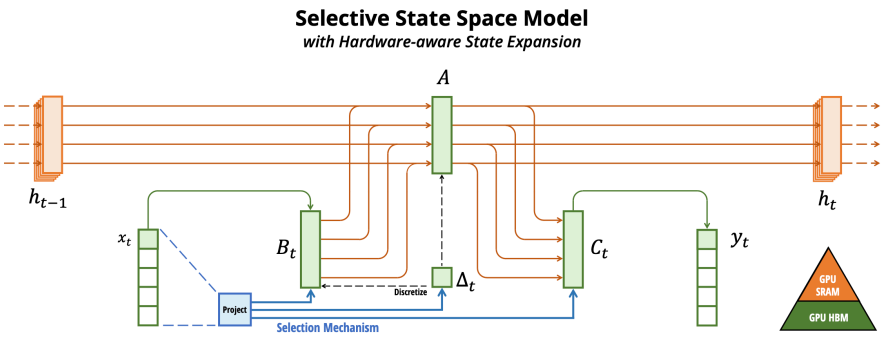


Figure 1: (**Overview**.) Structured SSMs independently map each channel (e.g. $D = 5$) of an input x to output y through a higher dimensional latent state h (e.g. $N = 4$). Prior SSMs avoid materializing this large effective state (DN, times batch size B and sequence length L) through clever alternate computation paths requiring time-invariance: the $(\Delta, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ parameters are constant across time. Our selection mechanism adds back input-dependent dynamics, which also requires a careful hardware-aware algorithm to only materialize the expanded states in more efficient levels of the GPU memory hierarchy.

Concretely, instead of preparing the scan input $(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}})$ of size $(\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N})$ in GPU HBM (high-bandwidth memory), we load the SSM parameters $(\Delta, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ directly from slow HBM to fast SRAM, perform the discretization and recurrence in SRAM, and then write the final outputs of size $(\mathtt{B}, \mathtt{L}, \mathtt{D})$ back to HBM.

Here, the following are fused into one kernel:

- Discretization step with *step size* Δ
- Selective scan algorithm
- Multiplication with C

4.2.3. Recomputation

During backpropagation, in order to calculate the gradients at each node, we need to cache the output values of the forward step. Since caching the activations and then reusing them during back-propagation means that we need to save them to the HBM and then copy them back from the HBM during back-propagation, it may be faster to just recompute them during backpropagation.

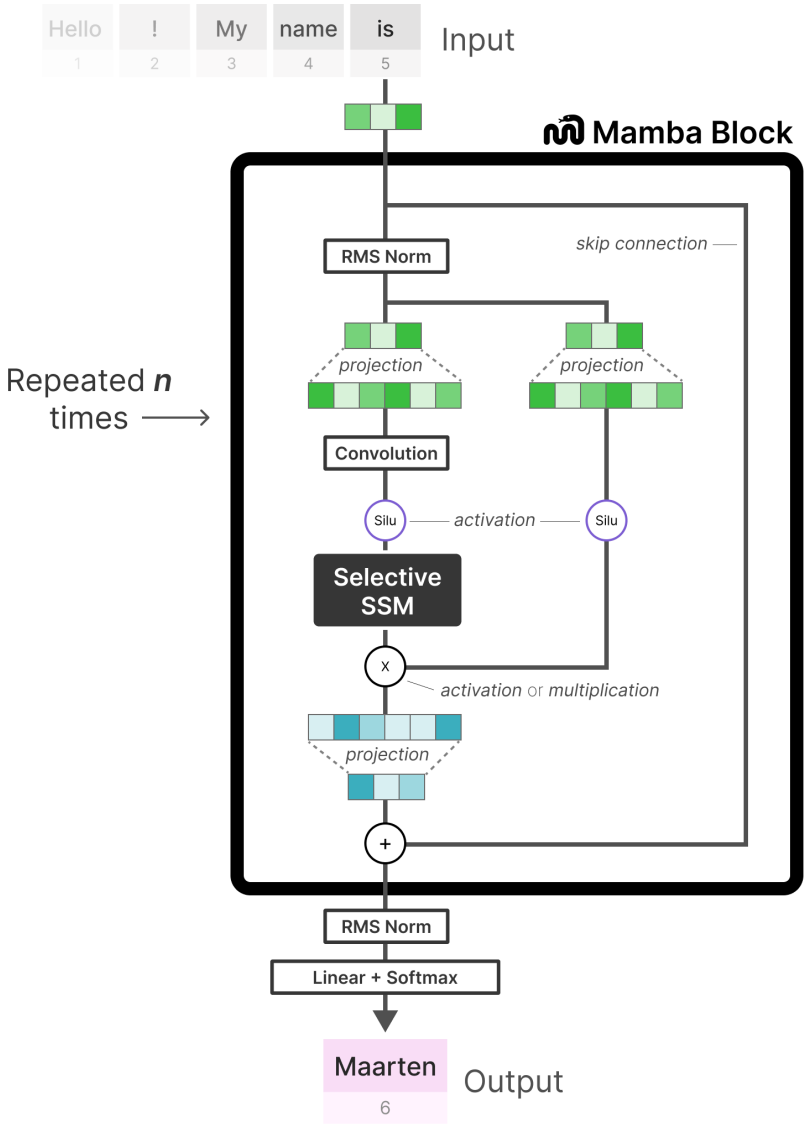
Finally, we must also avoid saving the intermediate states, which are necessary for backpropagation. We carefully apply the classic technique of recomputation to reduce the memory requirements: the intermediate states are not stored but recomputed in the backward pass when the inputs are loaded from HBM to SRAM. As a result, the fused selective scan layer has the same memory requirements as an optimized transformer implementation with FlashAttention.

4.3. Architecutre

- Mamba relaxes the time invariant property of S4
- SSMs are linear operation blocks, because of the linearity, backprop through time is reportedly more stable for Mamba than for e.g., LSTM (no activation function to squash signals to zero)
- There's nonlinearity in the outputs and in the gates.

- Convolution operation on expanded input emdeddings is to prevent the independent token calculation

Very similar to Decoder architecture, multiple Mamba blocks can be stacked as shown:



4.4. Benchmarks

4.4.1 Performance of Proposed Synthetic tasks

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

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

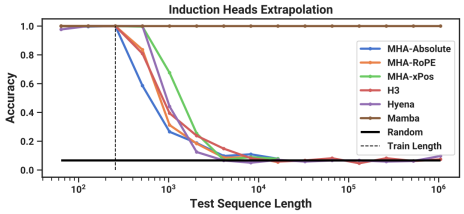


Table 2: (**Induction Heads**.) Models are trained on sequence length $2^8 = 256$, and tested on increasing sequence lengths of $2^6 = 64$ up to $2^{20} = 1048576$. Full numbers in Table 11.

4.4.2. Performance of Language Modelling (GPT3 reciepe & trained on Pile Dataset)

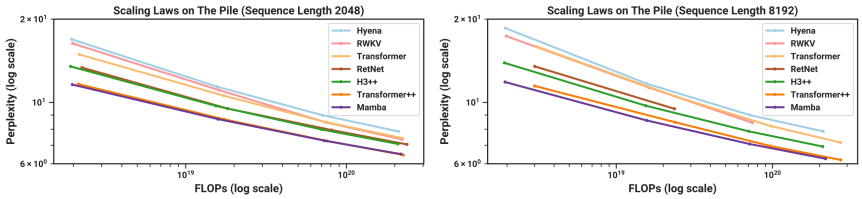


Figure 4: (**Scaling Laws**.) Models of size $\approx 125M$ to $\approx 1.3B$ parameters, trained on the Pile. Mamba scales better than all other attention-free models and is the first to match the performance of a very strong "Transformer++" recipe that has now become standard, particularly as the sequence length grows.

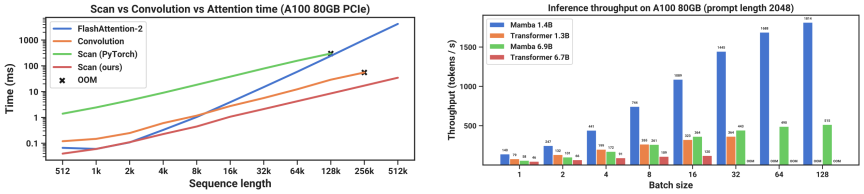
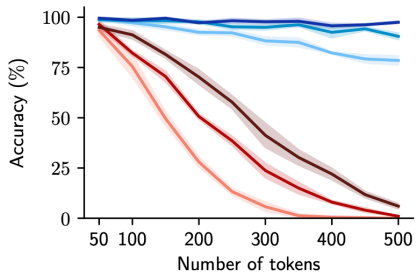


Figure 8: (**Efficiency Benchmarks**.) (*Left*) Training: our efficient scan is 40× faster than a standard implementation. (*Right*) Inference: as a recurrent model, Mamba can achieve 5× higher throughput than Transformers.

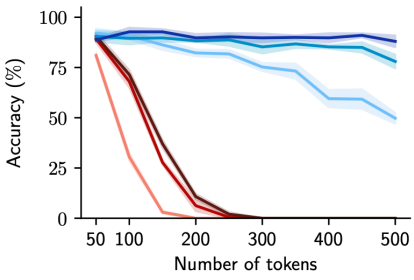
4.5. Short Comings

- Selective scan can overcome the weakness with descrete modalities (text, DNA) but this conversly can impede the performance on data that LTI system excels on (audio)
- Repeat After Me: Transformers are Better than State Space Models at Copying



Pythia: 410M 1.4B 2.8B
Mamba: 360M 1.4B 2.8B

(a) Copy: natural language strings



Pythia: 410M 1.4B 2.8B
Mamba: 360M 1.4B 2.8B

(b) Copy: shuffled strings

References Papers:

1. Mamba: Linear-Time Sequence Modeling with Selective State Spaces
2. Efficiently Modeling Long Sequences with Structured State Spaces
3. Simplified State Space Layers for Sequence Modeling
4. Repeat After Me: Transformers are Better than State Space Models at Copying

Thank You!