HiPPO

A memory efficient approach to state space models

Team Deep Space

- Deep Birenbhai Vaghasiya
- Naman Mishra
- Sanidhya Kaushik
- Sathvik Manthri

HiPPO: Recurrent Memory with Optimal Polynomial Projections

How can we remember context from millions of steps ago?

(How to represent entire cumulative history using bounded storage, which must be updated online as more data is received?)

Insight :

- Phrase memory as online function approx. where $f : \mathbb{R}_+ \rightarrow \mathbb{R}$ is summarized by storing its optimal coefficients in terms of some basis functions.
- This approx is evaluated w.r.t a measure that specifies the importance of each time in the past.

HiPPO : Higher-order Polynomial Projection Operators produces operators that project arbitrary functions onto the space of polynomials w.r.t a given measure.

HiPPO problem setup

Given an input function $f(t) \in \mathbb{R}$ on $t \geq 0$, many problems require operating on the cumulative *history* $f_{\leq t} := f(x) \mid_{x \leq t}$ 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.

- In order to fully specify this problem, we require two ingredients: a way to quantify the approximation and a suitable subspace.

HiPPO: Outline

- ➔ Formalizing the framework
- ➔ The HiPPO operators
- ➔ Discretization
- ➔ HiPPO in RNNs

General HiPPO framework

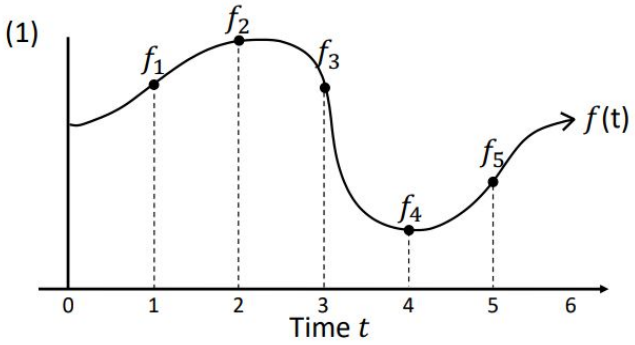
Given a time-varying measure family $\mu^{(t)}$ supported on $(-\infty, t]$, a sequence of basis functions $\mathcal{G} = \text{span}\{g_n^{(t)}\}_{n\in[N]}$, and a continuous function $f : \mathbb{R}_{\geq 0} \to \mathbb{R}$, HiPPO defines an operator that maps f to the optimal projection coefficients $c : \mathbb{R}_{\geq 0} \to \mathbb{R}^N$, such that

$$g^{(t)} := \text{argmin}_{g\in\mathcal{G}} \left\| f_{\leq t} - g \right\|_{\mu^{(t)}}, \qquad \text{and} \qquad g^{(t)} = \sum_{n=0}^{N-1} c_n(t) g_n^{(t)}.$$

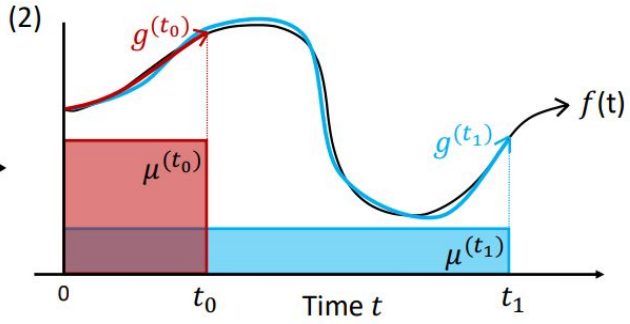
The first step refers to the proj_t operator and the second the coef_t operator.

We focus on the case where the coefficients $c(t)$ has the form of a linear ODE satisfying $\frac{d}{dt}c(t) = A(t)c(t) + B(t)f(t)$ for some $A(t) \in \mathbb{R}^{N\times N}$, $B(t) \in \mathbb{R}^{N\times 1}$.

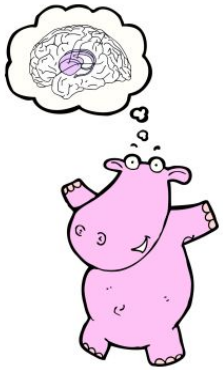
Illustration of HiPPO framework



proj_t



coef_t



(3)

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

Continuous-time HiPPO ODE

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

discretize

(4)

Discrete-time HiPPO Recurrence

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

Illustration of HiPPO measures

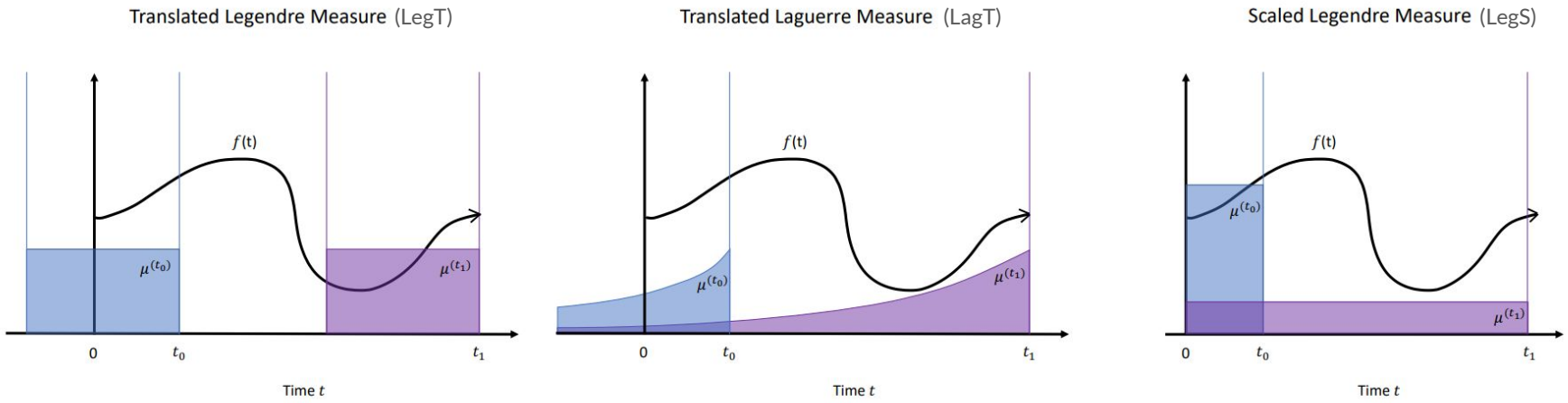


Figure : **Illustration of HiPPO measures.** At time t_0, the history of a function $f(x)_{x \leq t_0}$ is summarized by polynomial approximation with respect to the measure $\mu^{(t_0)}$ (blue), and similarly for time t_1 (purple). (Left) The Translated Legendre measure (**LegT**) assigns weight in the window $[t - \theta, t]$. For small t, $\mu^{(t)}$ is supported on a region $x < 0$ where f is not defined. When t is large, the measure is not supported near 0, causing the projection of f to forget the beginning of the function. (Middle) The Translated Laguerre (**LagT**) measure decays the past exponentially. It does not forget, but also assigns weight on $x < 0$. (Right) The Scaled Legendre measure (**LegS**) weights the entire history $[0, t]$ uniformly.

HiPPO operators

The HiPPO abstraction: online function approximation.

Definition *Given a time-varying measure family* $\mu^{(t)}$ *supported on* $(-\infty, t]$*, an* N*-dimensional subspace* $\mathcal{G}$ *of polynomials, and a continuous function* $f : \mathbb{R}_{\geq 0} \to \mathbb{R}$*, HiPPO defines a* projection *operator* proj_t *and a* coefficient extraction *operator* coef_t *at every time* t*, with the following properties:*

(1) proj_t *takes the function* f *restricted up to time* t*,* $f_{\leq t} := f(x) \mid_{x \leq t}$*, and maps it to a polynomial* $g^{(t)} \in \mathcal{G}$*, that minimizes the approximation error* $\| f_{\leq t} - g^{(t)} \|_{L_2(\mu^{(t)})}$.

(2) $\text{coef}_t : \mathcal{G} \to \mathbb{R}^N$ *maps the polynomial* $g^{(t)}$ *to the coefficients* $c(t) \in \mathbb{R}^N$ *of the basis of orthogonal polynomials defined with respect to the measure* $\mu^{(t)}$.

The composition $\text{coef} \circ \text{proj}$ *is called* hippo*, which is an operator mapping a function* $f : \mathbb{R}_{\geq 0} \to \mathbb{R}$ *to the optimal projection coefficients* $c : \mathbb{R}_{\geq 0} \to \mathbb{R}^N$*, i.e.* $(\text{hippo}(f))(t) = \text{coef}_t(\text{proj}_t(f))$.

Why ODE?

$$c_i(t) = \int f(x) P_i^{(t)}(x) \omega(t, x) \mathrm{d}x.$$

Differentiating this with respect to time, we get

$$\frac{\mathrm{d}}{\mathrm{d}t} c_i(t) = \int f(x) \frac{\partial}{\partial t} P_i^{(t)}(x) \omega(t, x) \mathrm{d}x + \int f(x) P_i^{(t)}(x) \frac{\partial}{\partial t} \omega(t, x) \mathrm{d}x.$$

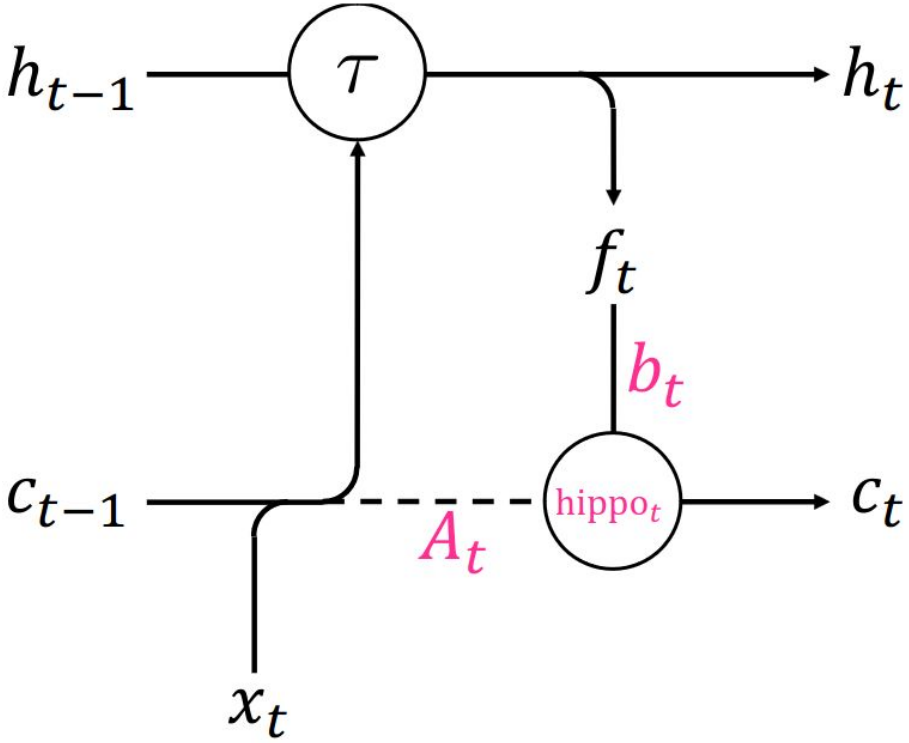
Discretization

- Hippo is a map on continuous functions
- As hippo defines a closed-form ODE of the coefficient dynamics, standard ODE discretization methods can be applied to turn this into discrete memory updates.
- Hippo, depending on the context, can refer to an ODE of form

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

or a recurrence

$$c_t = A_t c_{t-1} + B_t f_t,$$



$$
\begin{aligned}
h_t \in \mathbb{R}^d &= \tau(h_{t-1}, [c_{t-1}, x_t]) \\
f_t \in \mathbb{R}^1 &= \mathcal{L}_f(h_t) \\
c_t \in \mathbb{R}^N &= \text{hippo}_t(f) \\
&= A_t c_{t-1} + B_t f_t
\end{aligned}
$$

Figure : The simple RNN model we use HiPPO with, and associated update equations. $\mathcal{L}_\square$ is a parametrized linear function, τ is any RNN update function, and $[\cdot]$ denotes concatenation. hippo is the HiPPO memory operator which orthogonalizes the history of the f_t features up to time t. A_t, B_t are fixed matrices depending on the chosen measure . N and d represent the approximation order and hidden state size, respectively.

Theorem 1 *For LegT and LagT, the coefficients of the best polynomial approximation are given by linear time-invariant (LTI) ODEs* $\frac{\mathrm{d}}{\mathrm{d}t}c(t) = -Ac(t) + Bf(t)$*, where* $A \in \mathbb{R}^{N\times N}, B \in \mathbb{R}^{N\times 1}$*:*

LegT:

$$A_{nk} = \frac{1}{\theta}\begin{cases}(-1)^{n-k}(2n+1) & \text{if } n \geq k\\ 2n+1 & \text{if } n \leq k\end{cases}, \qquad B_n = \frac{1}{\theta}(2n+1)(-1)^n \tag{8}$$

LagT:

$$A_{nk} = \begin{cases}1 & \text{if } n \geq k\\ 0 & \text{if } n < k\end{cases}, \qquad B_n = 1 \tag{9}$$

Theorem 2 *The continuous-* (10) *and discrete-* (11) *time dynamics for* ***HiPPO-LegS*** *are:*

$$\frac{\mathrm{d}}{\mathrm{d}t}c(t) = -\frac{1}{t}Ac(t) + \frac{1}{t}Bf(t) \tag{10}$$

$$c_{k+1} - c_k = -\frac{A}{k}c_k + \frac{1}{k}Bf_k \tag{11}$$

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

Our Test Results for LegT, LagT and LegS on the permuted-MNIST dataset

Measure	Train accuracy	Test accuracy	Validation accuracy
LegT	96.9%	96.7%	96.1%
LagT	78.0%	79.1%	78.9%
LegS	98.6%	96.4%	96.1%

Table 1: Experiment results

Our methods and related baselines. Permuted MNIST (pMNIST) validation scores. (Top): Our methods. (Bottom): Recurrent baselines.

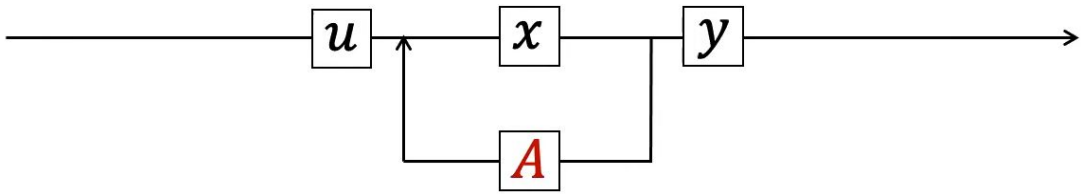
Method	Validation accuracy (%)
HiPPO-LegS	**98.34**
HiPPO-LagT $\Delta t = 1.0$	98.15
HiPPO-LegT $\theta = 200$	98.00
HiPPO-LegT $\theta = 2000$	97.90
HiPPO-LagT $\Delta t = 0.1$	96.44
HiPPO-LegT $\theta = 20$	91.75
HiPPO-LagT $\Delta t = 0.01$	90.71
HiPPO-Rand	69.93
LMU	97.08
ExpRNN	94.67
GRU	93.04
LSTM	92.54
MGU	89.37
RNN	52.98

SSM

The state space model is defined by this simple equation. It maps a 1-D input signal $u(t)$ to an N-D latent state $x(t)$ before projecting to a 1-D output signal $y(t)$.

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

Our goal is to simply use the SSM as a black-box representation in a deep sequence model, where $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ are parameters learned by gradient descent. For the remainder, we will omit the parameter $\boldsymbol{D}$ for exposition (or equivalently, assume $\boldsymbol{D} = 0$ because the term $\boldsymbol{D}u$ can be viewed as a skip connection and is easy to compute).

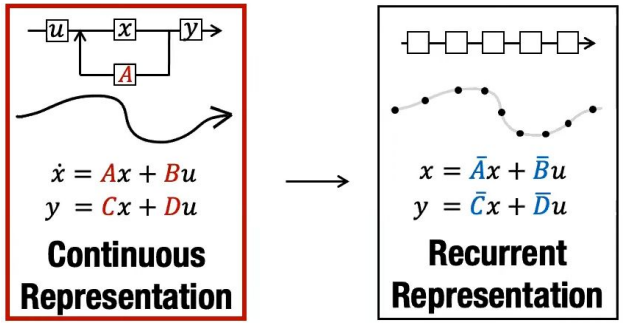


Computing with SSMs: Recurrent View

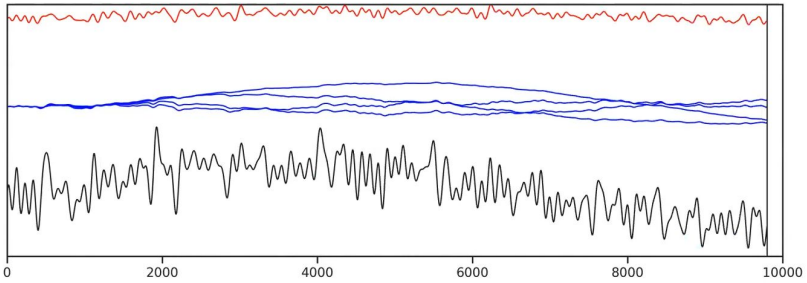
$$x'(t) = \boldsymbol{A}x(t) + \boldsymbol{B}u(t)$$
$$y(t) = \boldsymbol{C}x(t) + \boldsymbol{D}u(t)$$

To discretize the continuous-time SSM, we use the bilinear method, which converts the state matrix $\boldsymbol{A}$ into an approximation $\overline{\boldsymbol{A}}$. The discrete SSM is:

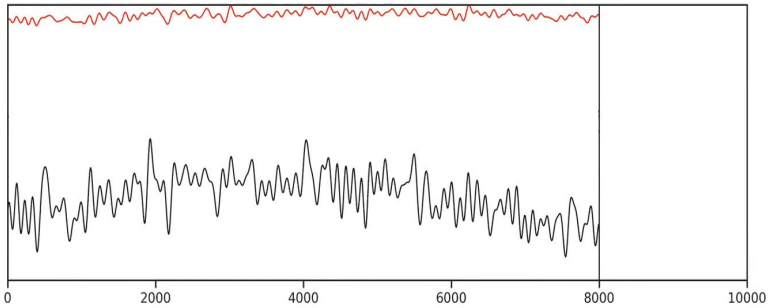
$$\overline{\boldsymbol{A}} = (\boldsymbol{I} - \Delta/2 \cdot \boldsymbol{A})^{-1}(\boldsymbol{I} + \Delta/2 \cdot \boldsymbol{A})$$
$$\overline{\boldsymbol{B}} = (\boldsymbol{I} - \Delta/2 \cdot \boldsymbol{A})^{-1}\Delta \boldsymbol{B}$$
$$\overline{\boldsymbol{C}} = \boldsymbol{C}$$



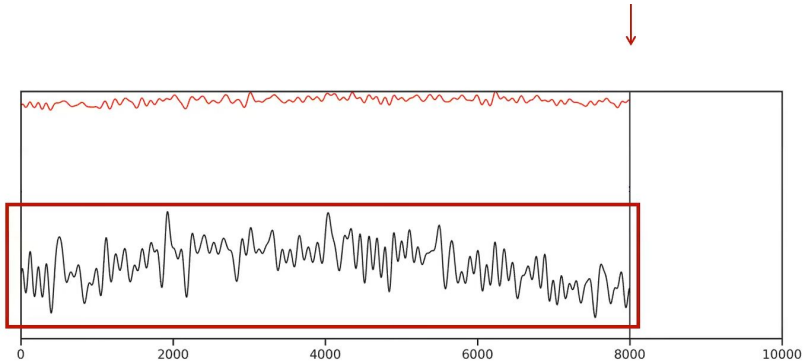
$$x_k = \overline{A}x_{k-1} + \overline{B}u_k \qquad y_k = \overline{C}x_k + \overline{D}u_k$$

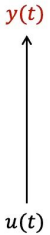


Efficient **autoregressive** computation of **state**



Next **output** depends on the **entire input**





...but can be computed in **constant** time

This equation is now a *sequence-to-sequence* map $u_k \mapsto y_k$ instead of function-to-function. Moreover the state equation is now a recurrence in x_k, allowing the discrete SSM to be computed like an RNN. Concretely, $x_k \in \mathbb{R}^N$ can be viewed as a *hidden state* with transition matrix $\overline{\boldsymbol{A}}$.

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

Can explicitly unroll the linear recurrence in closed form

$$x_0 = \overline{\boldsymbol{B}} u_0 \qquad x_1 = \overline{\boldsymbol{AB}} u_0 + \overline{\boldsymbol{B}} u_1 \qquad x_2 = \overline{\boldsymbol{A}}^2 \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{AB}} u_1 + \overline{\boldsymbol{B}} u_2 \qquad \dots$$
$$y_0 = \overline{\boldsymbol{CB}} u_0 \qquad y_1 = \overline{\boldsymbol{CAB}} u_0 + \overline{\boldsymbol{CB}} u_1 \qquad y_2 = \overline{\boldsymbol{CA}}^2 \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{CAB}} u_1 + \overline{\boldsymbol{CB}} u_2 \qquad \dots$$

- Every element of output sequence has a closed formula in terms of input. General form of output signal:

$$y_k = \overline{\boldsymbol{CA}}^k \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{CA}}^{k-1} \overline{\boldsymbol{B}} u_1 + \cdots + \overline{\boldsymbol{CAB}} u_{k-1} + \overline{\boldsymbol{CB}} u_k$$

- RNN's are hard to train as we have to do the training sequentially. But convolutions can be used where we can train parallely.

Training SSMs: The Convolutional Representation

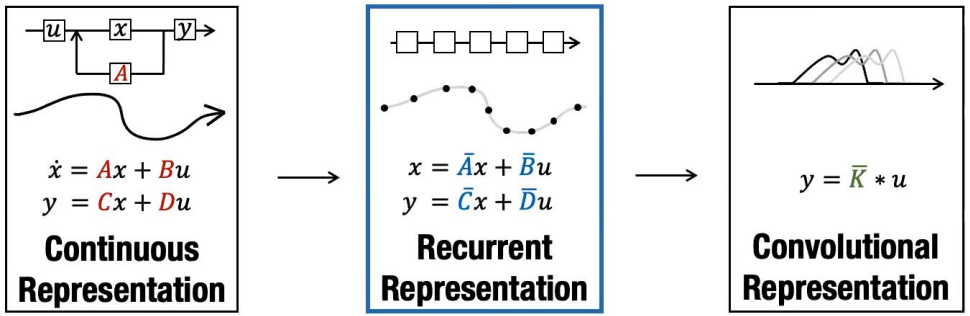
The punchline of this section is that we can turn the "RNN" above into a "CNN" by unrolling. Let's go through the derivation.

> The recurrent SSM is not practical for training on modern hardware due to its sequential nature. Instead, there is a well-known connection between linear time-invariant (LTI) SSMs and continuous convolutions. Correspondingly, the recurrent SSM can actually be written as a discrete convolution.

$$y_k = \overline{CA}^k \overline{B} u_0 + \overline{CA}^{k-1} \overline{B} u_1 + \cdots + \overline{CA}\,\overline{B} u_{k-1} + \overline{CB} u_k$$
$$y = \overline{K} * u$$

$$\overline{K} \in \mathbb{R}^L = (\overline{CB}, \overline{CAB}, \ldots, \overline{CA}^{L-1} \overline{B})$$

We call $\overline{K}$ the **SSM convolution kernel** or filter.



Structured State Space models (S4)

References

[1] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra and Christopher Ré. (2020) HiPPO: Recurrent Memory with Optimal Polynomial Projections.

[2] Albert Gu, Karan Goel and Christopher Ré. (2021) Efficiently Modeling Long Sequences with Structured State Spaces.

[3] Sasha Rush and Sidd Karamcheti. The Annotated S4 v3.

Thanks!