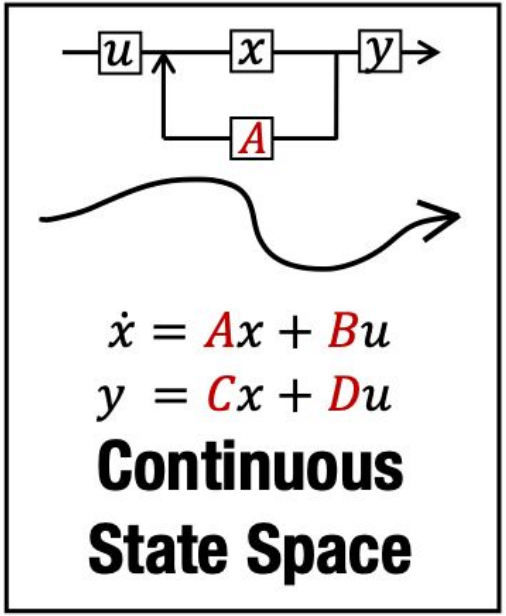
Structured state spaces

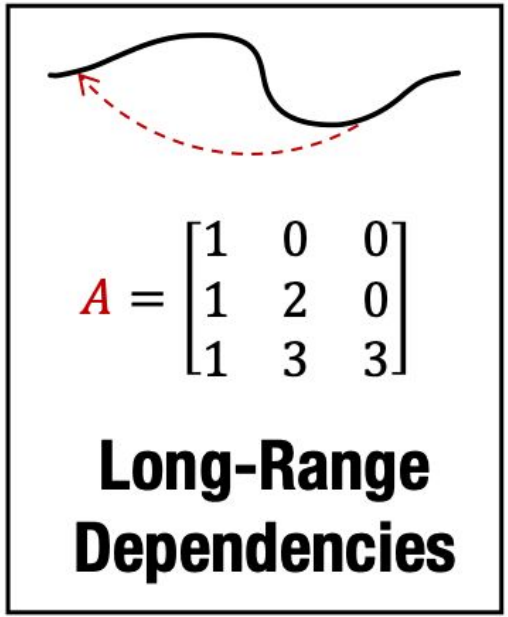
Albert Gu, Karan Goel, and Christopher Re

Different versions, same state space



Input u -> output y through latent x

Necessary for long-range dependencies



- Remember all previous inputs, but how?
- Special A matrix: HiPPO
  - We can't remember everything perfectly, projection onto low-dimensional subspace necessary

The HiPPo matrix for LRD

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%.

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

State space as an ODE

$$\dot{x}(t) = f(t, x(t))$$

$$x(t) = x(t_0) + \int_{t_0}^{t} f(s, x(s))\, ds.$$

Approximation:

$$x_{i+1}(t) := x_i(t_0) + \int_{t_0}^{t} f(s, x_i(s))\, ds$$

Discretization

$$x(t_{i+1}) = x(t_i) + \int_{t_i}^{t_{i+1}} f(s, x(s))\, ds$$

- There are a lot of different ways to solve this equation, many numerical solvers exist.
- Easiest: Euler's forward method:
  - **x(t+1) = x(t) + dt * dx/dt**
- Or, bilinear:

$$x(t + \Delta t) = (I - \alpha \Delta t \cdot A)^{-1}(I + (1 - \alpha)\Delta t \cdot A)x(t) + \Delta t(I - \alpha \Delta t \cdot A)^{-1} B \cdot u(t). \qquad (3)$$

$$\dot{x} = Ax + Bu$$

Recurrent representation of SSM

$$\boxed{\begin{aligned} x_k &= \overline{\boldsymbol{A}} x_{k-1} + \overline{\boldsymbol{B}} u_k \\ y_k &= \overline{\boldsymbol{C}} x_k \end{aligned}}$$

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

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

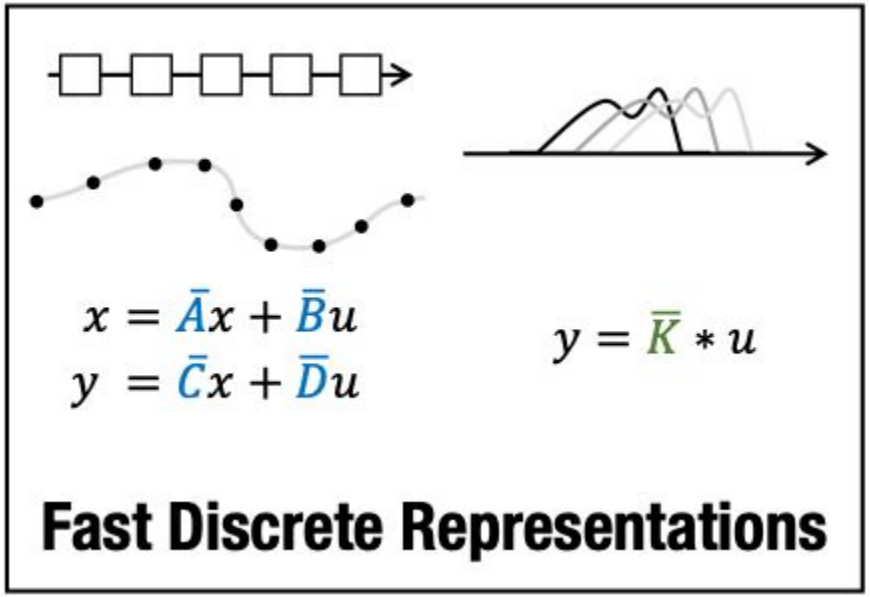
SSMs are gated RNNs

RNNs are LSSLs. We show two results about RNNs that may be of broader interest. Our first result says that the ubiquitous *gating mechanism* of RNNs, commonly perceived as a heuristic to smooth optimization [28], is actually the analog of a step size or timescale Δt.

Lemma 3.1. *A (1-D) gated recurrence* $x_t = (1 - \sigma(z))x_{t-1} + \sigma(z)u_t$*, where* σ *is the sigmoid function and* z *is an arbitrary expression, can be viewed as the GBT(*$\alpha = 1$*) (i.e.,* backwards-Euler*) discretization of a 1-D linear ODE* $\dot{x}(t) = -x(t) + u(t)$.

Proof. Applying a discretization requires a positive step size Δt. The simplest way to parameterize a positive function is via the exponential function $\Delta t = \exp(z)$ applied to any expression z. Substituting this into (3) with $A = -1, B = 1, \alpha = 1$ exactly produces the gated recurrence. □

State space models have a convolutional rep.



State space models have a convolutional rep.

$$x_0 = \overline{\boldsymbol{B}} u_0$$

$$y_0 = \overline{\boldsymbol{C}\boldsymbol{B}} u_0$$

$$x_1 = \overline{\boldsymbol{A}\boldsymbol{B}} u_0 + \overline{\boldsymbol{B}} u_1$$

$$y_1 = \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}} u_0 + \overline{\boldsymbol{C}\boldsymbol{B}} u_1$$

State space models have a convolutional rep.

$$x_1 = \overline{\boldsymbol{AB}} u_0 + \overline{\boldsymbol{B}} u_1$$

$$y_1 = \overline{\boldsymbol{CAB}} u_0 + \overline{\boldsymbol{CB}} u_1$$

$$x_2 = \overline{\boldsymbol{A}}^2 \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{AB}} u_1 + \overline{\boldsymbol{B}} u_2$$

$$y_2 = \overline{\boldsymbol{CA}}^2 \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{CAB}} u_1 + \overline{\boldsymbol{CB}} u_2$$

State space models have a convolutional rep.

$$y_k = \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^k\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{k-1}\overline{\boldsymbol{B}}u_1 + \cdots + \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}u_{k-1} + \overline{\boldsymbol{C}}\overline{\boldsymbol{B}}u_k$$
$$y = \overline{\boldsymbol{K}} * u.$$

Krylov subspace (used for power iteration factorization to find largest eigenvalues)

$$\mathcal{K}_r(A, b) = \operatorname{span}\{b, Ab, A^2b, \ldots, A^{r-1}b\}.$$

$$\overline{\boldsymbol{K}} \in \mathbb{R}^L := \mathcal{K}_L(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}) := \left(\overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^i\overline{\boldsymbol{B}}\right)_{i\in[L]} = (\overline{\boldsymbol{C}}\overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}, \ldots, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}}).$$

What is the intuition behind this?

In the particular case of LSSLs with HiPPO matrices (Sections 2 and 4.1), there is another intuitive interpretation of how LSSL relate to convolutions. Consider the special case when A corresponds to a uniform measure (in the literature known as the LMU [58] or HiPPO-LegT [24] matrix). Then for a fixed dt, equation (1) is simply memorizing the input within sliding windows of $\frac{1}{\Delta t}$ elements, and equation (2) extracts features from this window. Thus the LSSL can be interpreted as automatically learning convolution filters with a learnable kernel width.

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{1}$$

$$y(t) = Cx(t) + Du(t), \tag{2}$$

Cool! What now?

The fundamental bottleneck in computing the discrete-time SSM (3) is that it involves repeated matrix multiplication by $\overline{\boldsymbol{A}}$. For example, computing (5) naively as in the LSSL involves L successive multiplications by $\overline{\boldsymbol{A}}$, requiring $O(N^2L)$ operations and $O(NL)$ space.

Diagonalization may be the answer

Lemma 3.1. *Conjugation is an equivalence relation on SSMs* $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) \sim (\boldsymbol{V}^{-1}\boldsymbol{A}\boldsymbol{V}, \boldsymbol{V}^{-1}\boldsymbol{B}, \boldsymbol{C}\boldsymbol{V})$.

Proof. Write out the two SSMs with state denoted by x and $\tilde{x}$ respectively:

$$x' = \boldsymbol{A}x + \boldsymbol{B}u \qquad \tilde{x}' = \boldsymbol{V}^{-1}\boldsymbol{A}\boldsymbol{V}\tilde{x} + \boldsymbol{V}^{-1}\boldsymbol{B}u$$
$$y = \boldsymbol{C}x \qquad y = \boldsymbol{C}\boldsymbol{V}\tilde{x}$$

After multiplying the right side SSM by $\boldsymbol{V}$, the two SSMs become identical with $x = \boldsymbol{V}\tilde{x}$. Therefore these compute the exact same operator $u \mapsto y$, but with a change of basis by $\boldsymbol{V}$ in the state x. □

Diagonalization may be the answer

Lemma 3.1 motivates putting $\boldsymbol{A}$ into a canonical form by conjugation[2], which is ideally more structured and allows faster computation. For example, if $\boldsymbol{A}$ were diagonal, the resulting computations become much more tractable. In particular, the desired $\overline{\boldsymbol{K}}$ (equation (4)) would be a **Vandermonde product** which theoretically only needs $O((N+L)\log^2(N+L))$ arithmetic operations [29].

$$V = V(x_0, x_1, \cdots, x_m) = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^n \end{bmatrix}$$

How do we diagonalize the HiPPO matrix

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

Numerical issues

Lemma 3.2. *The HiPPO matrix* $\boldsymbol{A}$ *in equation* (2) *is diagonalized by the matrix* $\boldsymbol{V}_{ij} = \binom{i+j}{i-j}$. *In particular,* $\boldsymbol{V}_{3i,i} = \binom{4i}{2i} \approx 2^{4i}$. *Therefore* $\boldsymbol{V}$ *has entries of magnitude up to* $2^{4N/3}$.

How do we avoid numerical issues?

The previous discussion implies that we should only conjugate by well-conditioned matrices $\boldsymbol{V}$. The ideal scenario is when the matrix $\boldsymbol{A}$ is diagonalizable by a perfectly conditioned (i.e., unitary) matrix. By the Spectral Theorem of linear algebra, this is exactly the class of **normal matrices**. However, this class of matrices is restrictive; in particular, it does not contain the HiPPO matrix (2).

Decomposition

We make the observation that although the HiPPO matrix is not normal, it can be decomposed as the *sum of a normal and low-rank matrix*. However, this is still not useful by itself: unlike a diagonal matrix, powering up this sum (in (5)) is still slow and not easily optimized. We overcome this bottleneck by simultaneously applying three new techniques.

Decomposition

Theorem 1. *All HiPPO matrices from [16] have a NPLR representation*

$$\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^* - \boldsymbol{P}\boldsymbol{Q}^\top = \boldsymbol{V}\left(\boldsymbol{\Lambda} - (\boldsymbol{V}^*\boldsymbol{P})(\boldsymbol{V}^*\boldsymbol{Q})^*\right)\boldsymbol{V}^* \tag{6}$$

for unitary $\boldsymbol{V} \in \mathbb{C}^{N\times N}$*, diagonal* $\boldsymbol{\Lambda}$*, and low-rank factorization* $\boldsymbol{P}, \boldsymbol{Q} \in \mathbb{R}^{N\times r}$*. These matrices HiPPO- LegS, LegT, LagT all satisfy* $r = 1$ *or* $r = 2$*. In particular, equation (2) is NPLR with* $r = 1$*.*

Full algorithm

Algorithm 1 S4 Convolution Kernel (Sketch)

Input: S4 parameters $\boldsymbol{\Lambda}, \boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{C}^N$ and step size Δ

Output: SSM convolution kernel $\overline{\boldsymbol{K}} = \mathcal{K}_L(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}})$ for $\boldsymbol{A} = \boldsymbol{\Lambda} - \boldsymbol{P}\boldsymbol{Q}^*$ (equation (5))

1: $\widetilde{\boldsymbol{C}} \leftarrow \left(\boldsymbol{I} - \overline{\boldsymbol{A}}^L\right)^* \overline{\boldsymbol{C}}$ ▷ Truncate SSM generating function (SSMGF) to length L

2: $\begin{bmatrix} k_{00}(\omega) & k_{01}(\omega) \\ k_{10}(\omega) & k_{11}(\omega) \end{bmatrix} \leftarrow \left[\widetilde{\boldsymbol{C}}\ \boldsymbol{Q}\right]^* \left(\frac{2}{\Delta}\frac{1-\omega}{1+\omega} - \boldsymbol{\Lambda}\right)^{-1} [\boldsymbol{B}\ \boldsymbol{P}]$ ▷ Black-box Cauchy kernel

3: $\hat{\boldsymbol{K}}(\omega) \leftarrow \frac{2}{1+\omega}\left[k_{00}(\omega) - k_{01}(\omega)(1 + k_{11}(\omega))^{-1} k_{10}(\omega)\right]$ ▷ Woodbury Identity

4: $\hat{\boldsymbol{K}} = \{\hat{\boldsymbol{K}}(\omega) : \omega = \exp(2\pi i \frac{k}{L})\}$ ▷ Evaluate SSMGF at all roots of unity $\omega \in \Omega_L$

5: $\overline{\boldsymbol{K}} \leftarrow \mathsf{iFFT}(\hat{\boldsymbol{K}})$ ▷ Inverse Fourier Transform

- Instead of computing $\overline{\boldsymbol{K}}$ directly, we compute its spectrum by evaluating its **truncated generating function** $\sum_{j=0}^{L-1} \overline{\boldsymbol{K}}_j \zeta^j$ at the roots of unity ζ. $\overline{\boldsymbol{K}}$ can then be found by applying an inverse FFT.
- This generating function is closely related to the matrix resolvent, and now involves a matrix *inverse* instead of *power*. The low-rank term can now be corrected by applying the **Woodbury identity** which reduces $(\boldsymbol{A} + \boldsymbol{P}\boldsymbol{Q}^*)^{-1}$ in terms of $\boldsymbol{A}^{-1}$, truly reducing to the diagonal case.
- Finally, we show that the diagonal matrix case is equivalent to the computation of a **Cauchy kernel** $\frac{1}{\omega_j - \zeta_k}$, a well-studied problem with stable near-linear algorithms [30, 31].

Improvement

Finally, we note that follow-up work found that this version of S4 can sometimes suffer from numerical instabilities when the $\boldsymbol{A}$ matrix has eigenvalues on the right half-plane [14]. It introduced a slight change to the NPLR parameterization for S4 from $\boldsymbol{\Lambda} - \boldsymbol{P}\boldsymbol{Q}^*$ to $\boldsymbol{\Lambda} - \boldsymbol{P}\boldsymbol{P}^*$ that corrects this potential problem.

Results: efficiency

Table 2: Deep SSMs: The S4 parameterization with Algorithm 1 is asymptotically more efficient than the LSSL.

	Training Step (ms)			Memory Alloc. (MB)		
Dim.	128	256	512	128	256	512
LSSL	9.32	20.6	140.7	222.1	1685	13140
S4	4.77	3.07	4.75	5.3	12.6	33.5
Ratio	1.9×	6.7×	**29.6**×	42.0×	133×	**392**×

Table 3: Benchmarks vs. efficient Transformers

	Length 1024		Length 4096	
	Speed	Mem.	Speed	Mem.
Transformer	1×	1×	1×	1×
Performer	1.23×	0.43×	3.79×	0.086×
Linear Trans.	**1.58**×	**0.37**×	**5.35**×	**0.067**×
S4	**1.58**×	0.43×	5.19×	0.091×

Results: long range arena

Table 4: (**Long Range Arena**) (*Top*) Original Transformer variants in LRA. Full results in Appendix D.2. (*Bottom*) Other models reported in the literature. *Please read Appendix D.5 before citing this table.*

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Transformer	36.37	64.27	57.46	42.44	71.40	✗	53.66
Reformer	37.27	56.10	53.40	38.07	68.50	✗	50.56
BigBird	36.05	64.02	59.29	40.83	74.87	✗	54.17
Linear Trans.	16.13	65.90	53.09	42.34	75.30	✗	50.46
Performer	18.01	65.40	53.82	42.77	77.05	✗	51.18
FNet	35.33	65.11	59.61	38.67	77.80	✗	54.42
Nyströmformer	37.15	65.52	79.56	41.58	70.94	✗	57.46
Luna-256	37.25	64.57	79.29	47.38	77.72	✗	59.37
S4	**59.60**	**86.82**	**90.90**	**88.65**	**94.20**	**96.35**	**86.09**

Interpretability

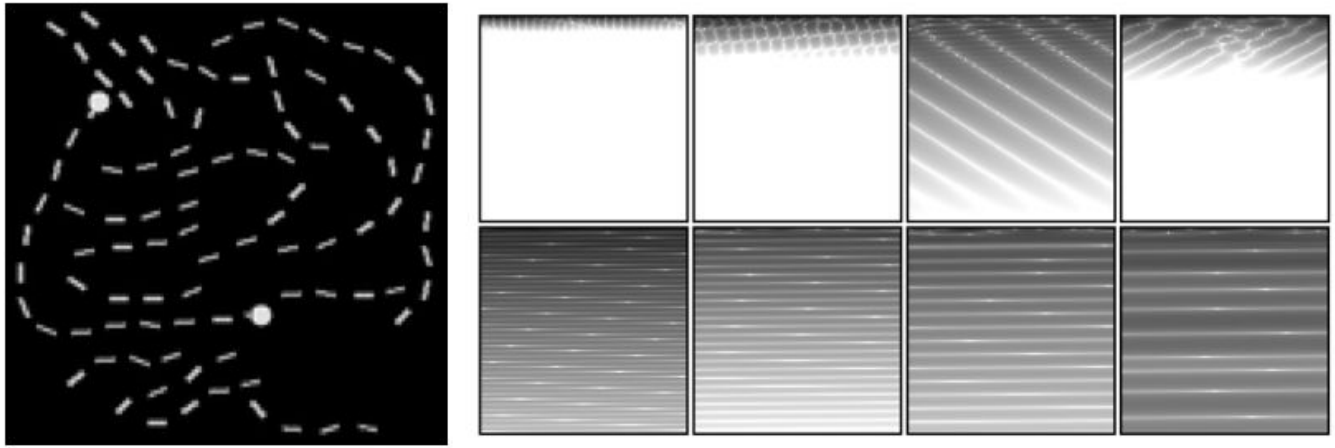


Figure 2: Visualizations of a trained S4 model on LRA Path-X. SSM convolution kernels $\overline{\boldsymbol{K}} \in \mathbb{R}^{16384}$ are reshaped into a 128×128 image. (*Left*) Example from the Path-X task, which involves deducing if the markers are connected by a path (*Top*) Filters from the first layer (*Bottom*) Filters from the last layer.

Results: sequence models

Table 5: (**SC10 classification**) Transformer, CTM, RNN, CNN, and SSM models. (*MFCC*) Standard preprocessed MFCC features (length 161). (*Raw*) Unprocessed signals (length 16000). (*0.5×*) Frequency change at test time. ✗ denotes not applicable or computationally infeasible on single GPU. *Please read Appendix D.5 before citing this table.*

	MFCC	RAW	0.5×
Transformer	90.75	✗	✗
Performer	80.85	30.77	30.68
ODE-RNN	65.9	✗	✗
NRDE	89.8	16.49	15.12
ExpRNN	82.13	11.6	10.8
LipschitzRNN	88.38	✗	✗
CKConv	**95.3**	71.66	65.96
WaveGAN-D	✗	96.25	✗
LSSL	93.58	✗	✗
S4	93.96	**98.32**	**96.30**

Table 6: (**Pixel-level 1-D image classification**) Comparison against reported test accuracies from prior works (Transformer, RNN, CNN, and SSM models). Extended results and citations in Appendix D.

	sMNIST	pMNIST	sCIFAR
Transformer	98.9	97.9	62.2
LSTM	98.9	95.11	63.01
r-LSTM	98.4	95.2	72.2
UR-LSTM	99.28	96.96	71.00
UR-GRU	99.27	96.51	74.4
HiPPO-RNN	98.9	98.3	61.1
LMU-FFT	-	98.49	-
LipschitzRNN	99.4	96.3	64.2
TCN	99.0	97.2	-
TrellisNet	99.20	98.13	73.42
CKConv	99.32	98.54	63.74
LSSL	99.53	**98.76**	84.65
S4	**99.63**	98.70	**91.13**

Results: forecasting

Table 9: Univariate long sequence time-series forecasting results. Full results in Appendix D.3.5.

	S4		Informer		LogTrans		Reformer		LSTMa		DeepAR		ARIMA		Prophet	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
$ETTh_1$	**0.116**	**0.271**	0.269	0.435	0.273	0.463	2.112	1.436	0.683	0.768	0.658	0.707	0.659	0.766	2.735	3.253
$ETTh_2$	**0.187**	**0.358**	0.277	0.431	0.303	0.493	2.030	1.721	0.640	0.681	0.429	0.580	2.878	1.044	3.355	4.664
$ETTm_1$	**0.292**	**0.466**	0.512	0.644	0.598	0.702	1.793	1.528	1.064	0.873	2.437	1.352	0.639	0.697	2.747	1.174
Weather	**0.245**	**0.375**	0.359	0.466	0.388	0.499	2.087	1.534	0.866	0.809	0.499	0.596	1.062	0.943	3.859	1.144
ECL	**0.432**	**0.497**	0.582	0.608	0.624	0.645	7.019	5.105	1.545	1.006	0.657	0.683	1.370	0.982	6.901	4.264

Results: Importance of HiPPO

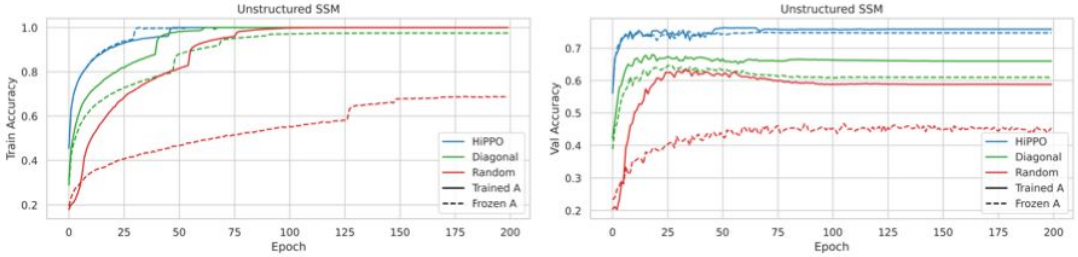
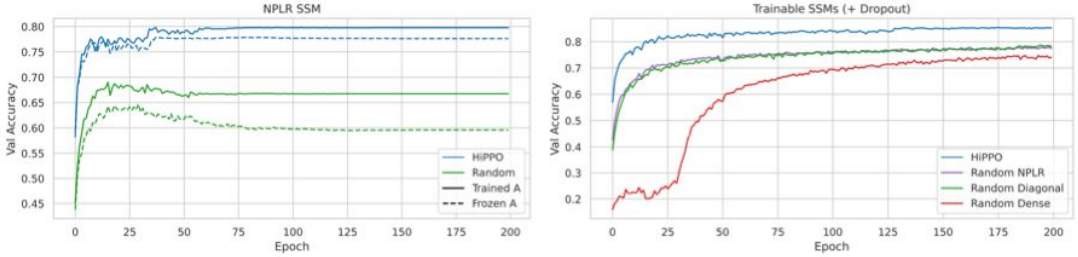


Figure 3: CIFAR-10 classification with unconstrained, real-valued SSMs with various initializations. (*Left*) Train accuracy. (*Right*) Validation accuracy.



Discussion