Brief Introduction to S4 Model

Eric Qu

zq32@duke.edu

Class of 2023
Duke Kunshan University

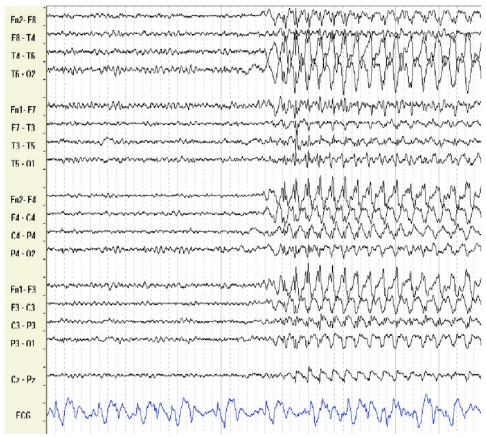
August 12, 2022

Intro

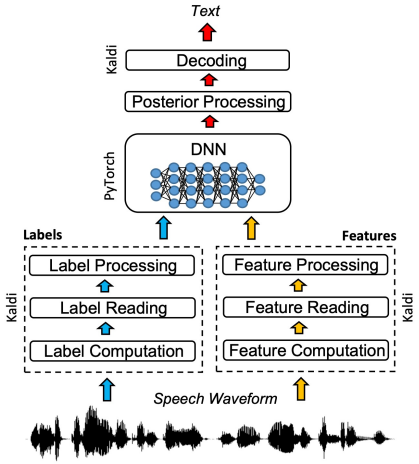
- In this talk, I will be introducing a new sequence model
- Related papers:
- HiPPO: Recurrent Memory with Optimal Polynomial Projections [1] (NIPS 2020 Spotlight)
- Efficiently Modeling Long Sequences with Structured State Spaces [2] (ICLR 2022 Oral)
- From Stanford, same lab as HGCN

Long "Continuous" Time Series

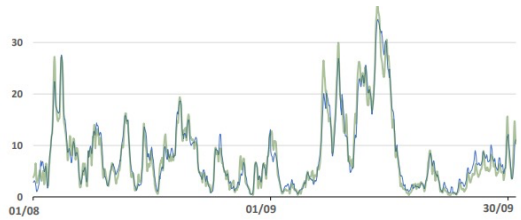
- Time Series sampled from a continuous physical process



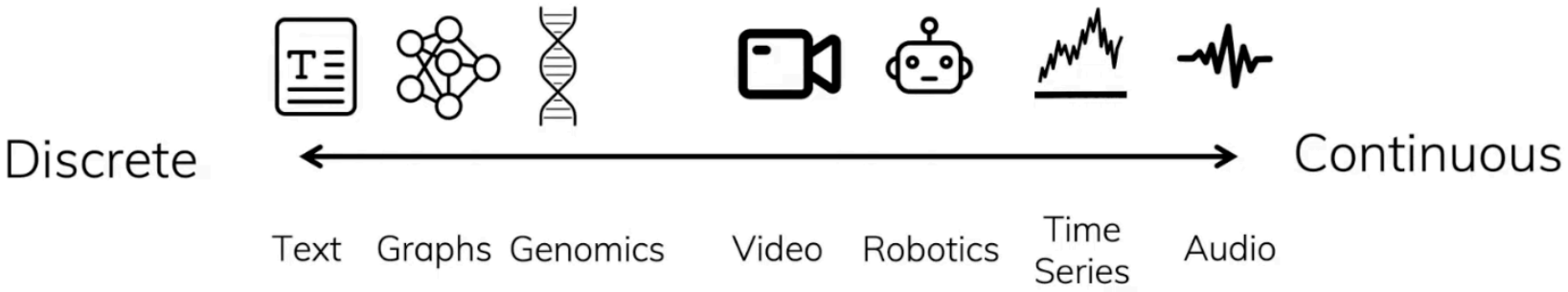
EEG/ECG



Speech



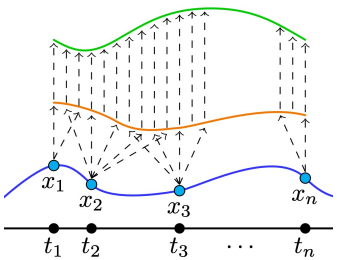
Energy Forecasting



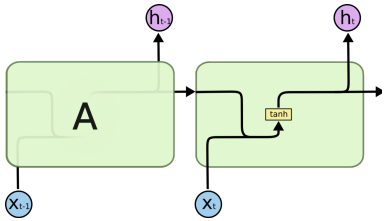
Sequence Modeling Paradigms

- Deep Sequence Models
- Sequence-to-sequence map
- (batch, length, dim) $\rightarrow$ (batch, length, dim)

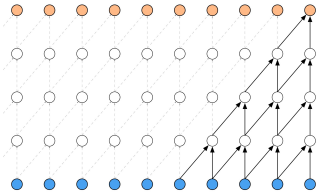
Continuous-time Model



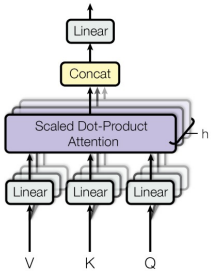
Recurrent Neural Net.



Convolutional Neural Net.

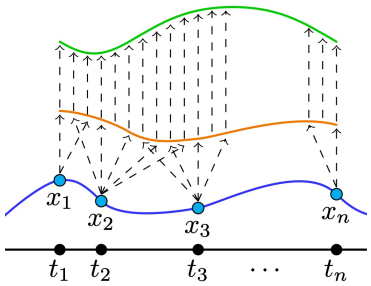


Transformer



Sequence Modeling Paradigms

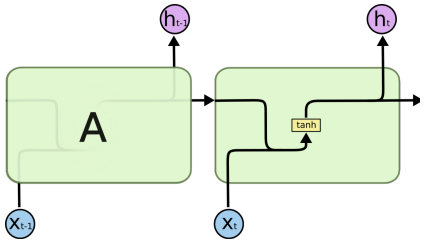
- Existing model families have clear tradeoffs
- All struggle with long-range dependencies (LRD)



✓ Continuous data
Irregular sampling

✗ Complex, very inefficient
Vanishing gradients

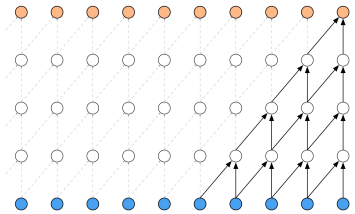
Continuous-time (CTM)



✓ Unbounded context
Stateful inference

✗ Inefficient training
Vanishing gradients

Recurrent (RNN)



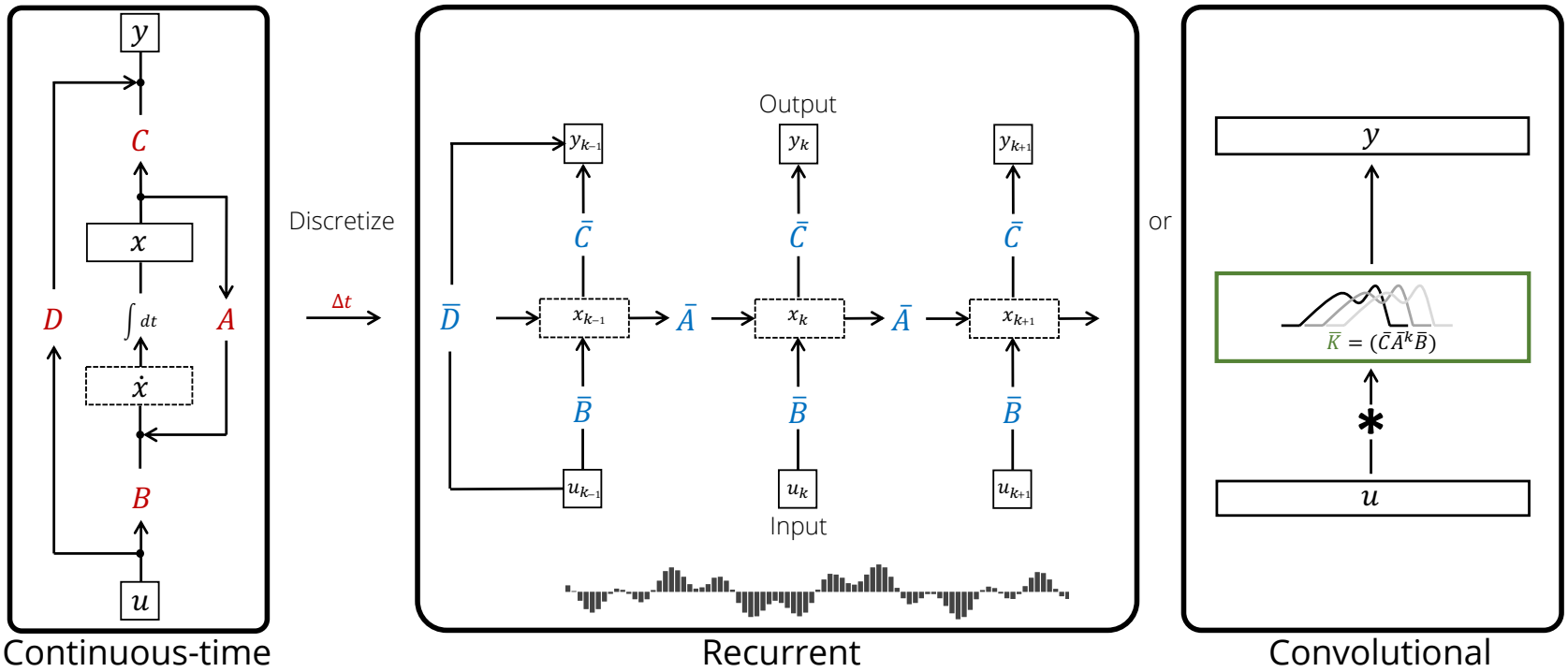
✓ Easy optimization
Parallelizable training

✗ Inefficient inference
Bounded context

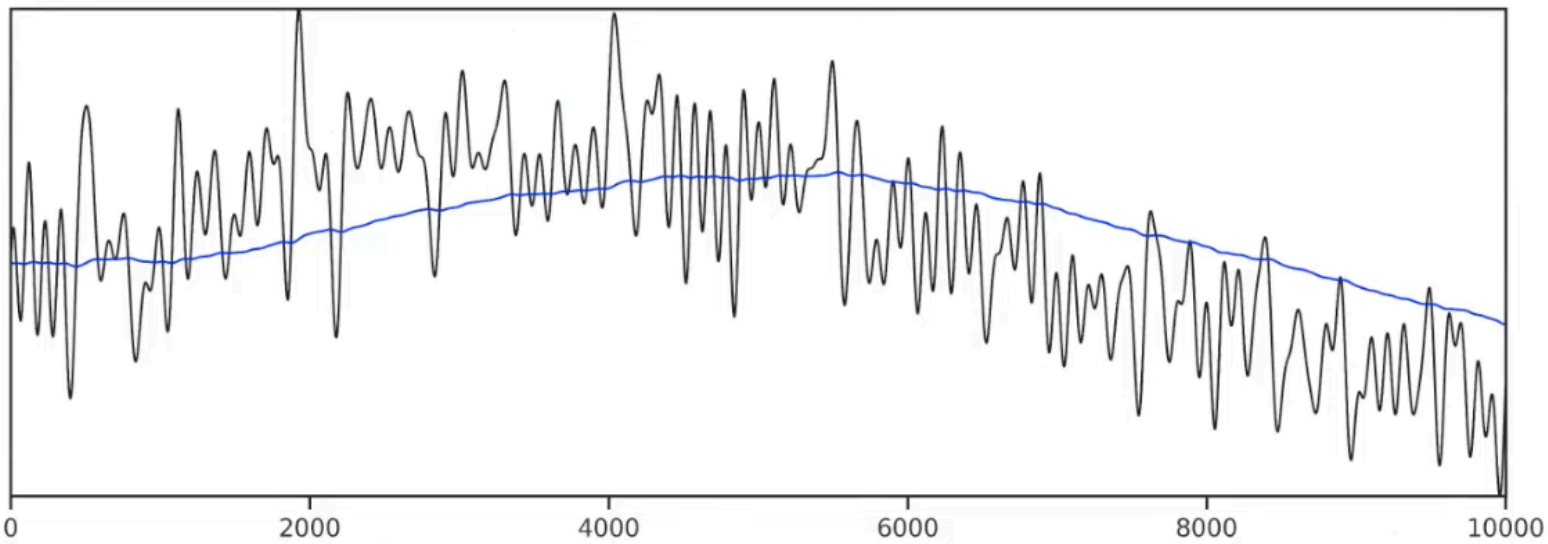
Convolutional (CNN)

Sequence Modeling Paradigms

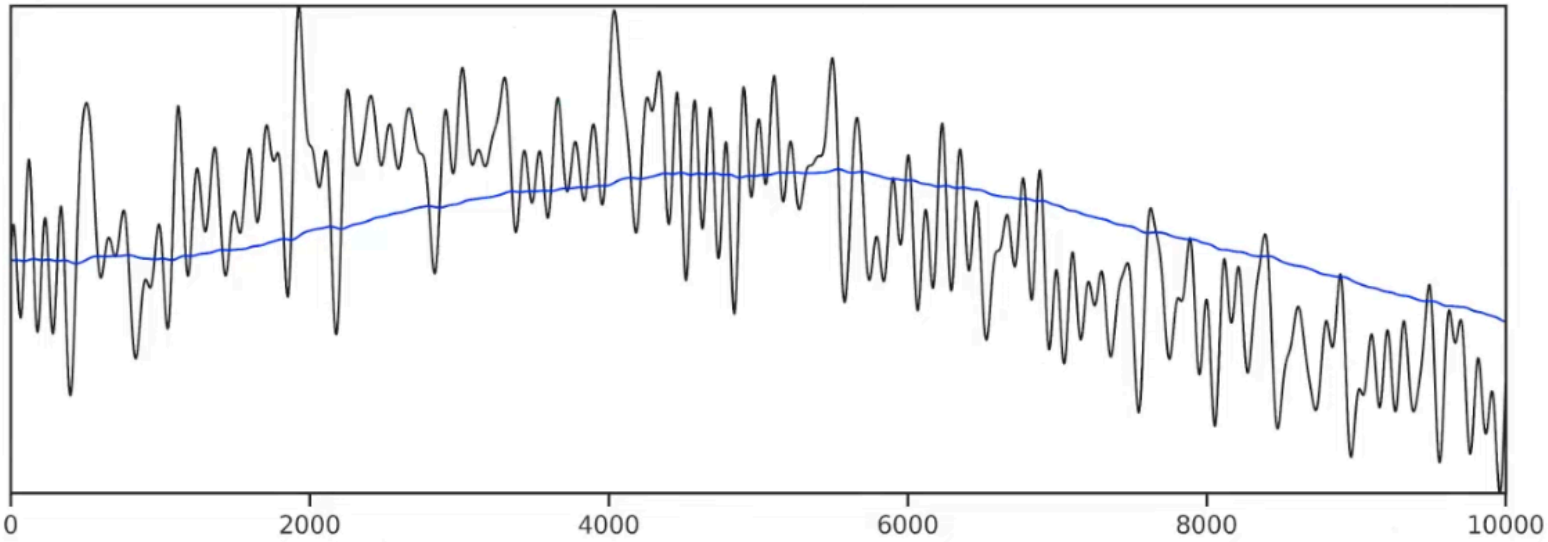
- S4 could take advantage of all three models



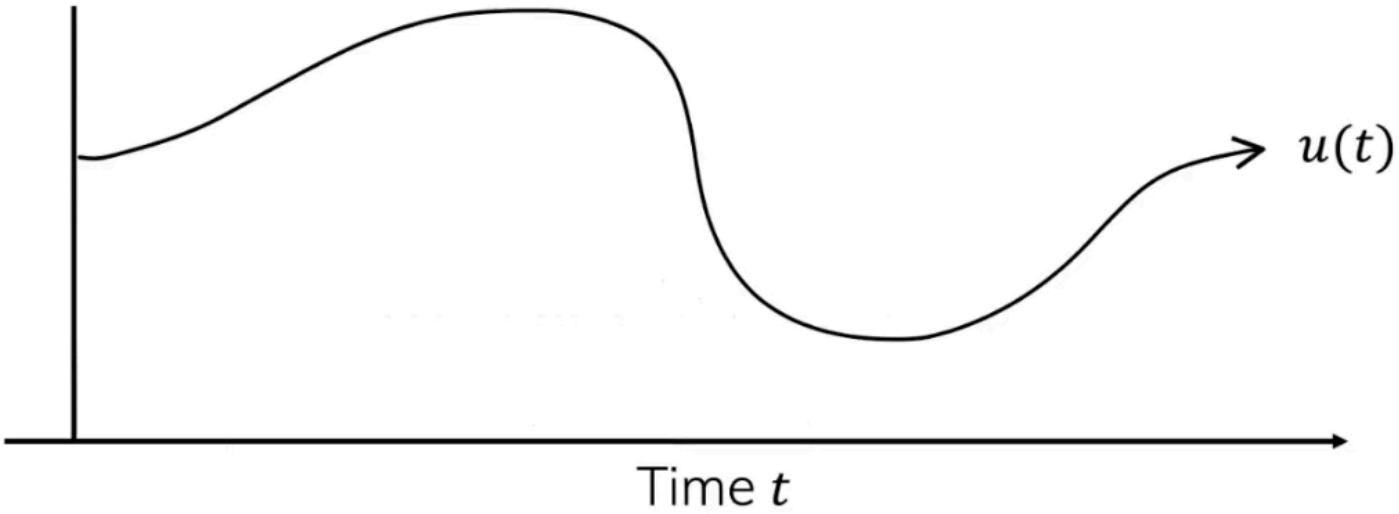
- To introduce S4, we need to mention a pervious work: HiPPO
- Motivation: Exponential Moving Average (EMA)
- A common method in feature engineering
- Modern models (Attention/Convolution) could not learn this
  - EMA has unbounded context (weighted average of all history)
  - RNN is better, but also not well due to vanishing gradients



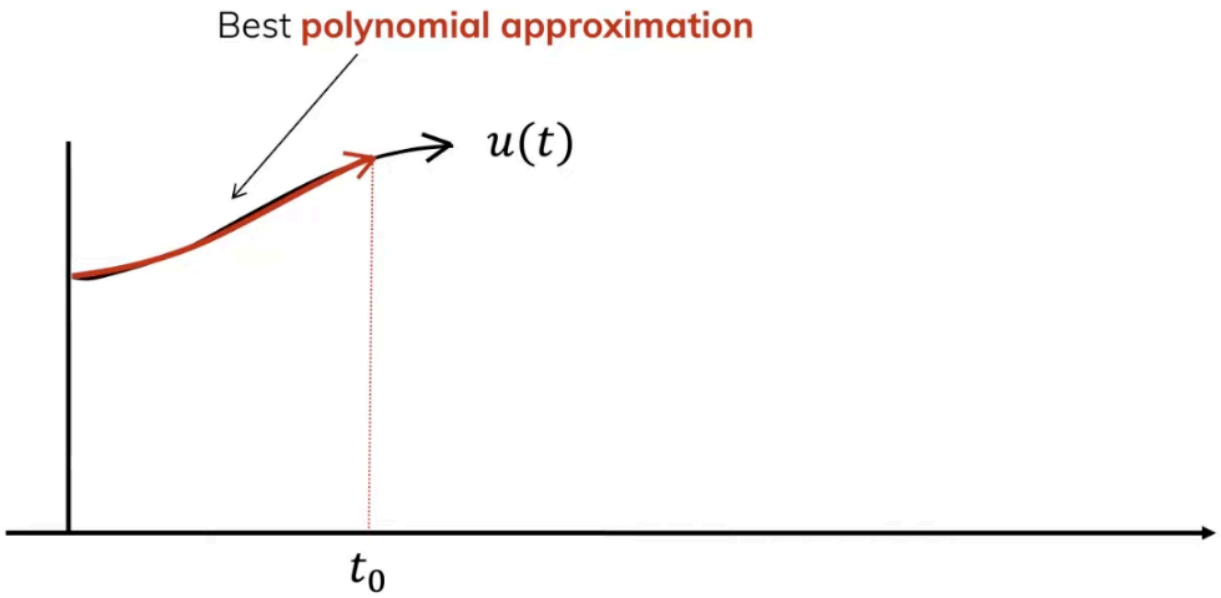
- Properties of EMA
  - Summary of the entire context
  - Update in constant time
- We want
  - Compress history $\rightarrow$ Reconstruct history (Unbounded context)
  - Observe a function online $\rightarrow$ maintain best reconstruction



- Observe input signal online
- Maintain compressed representation given memory budget

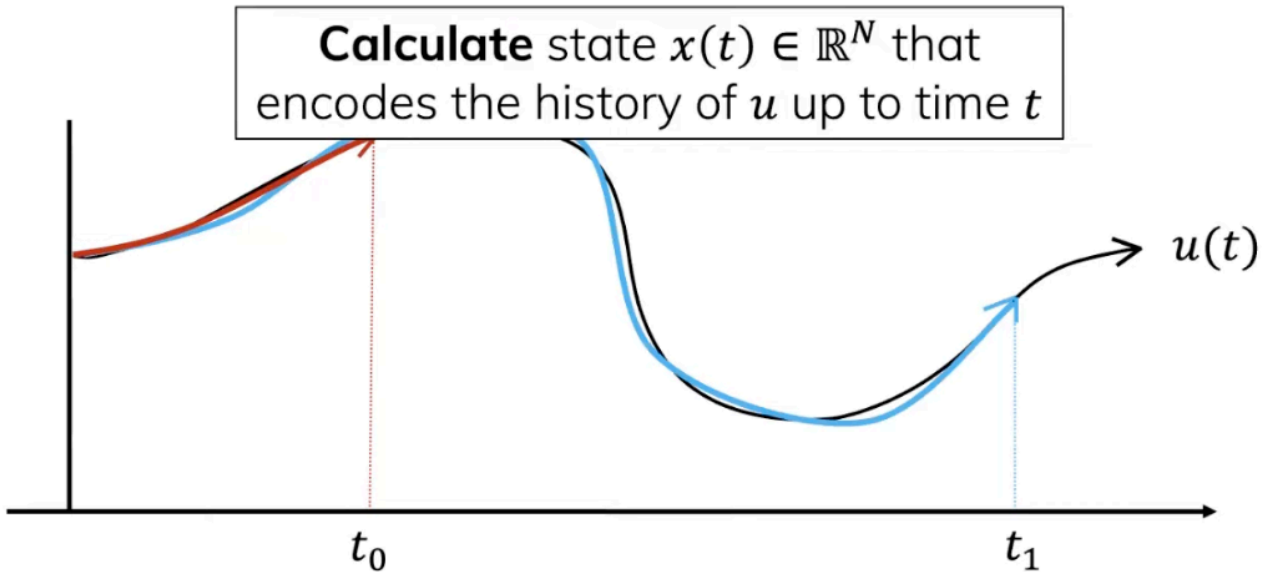


- At initial time t_0, we observe a part of the function
- We want to store the best approximation
  - For example, store the best polynomial approximation
  - Degree of the polynomial $\Leftrightarrow$ Memory budget



$$x(t_0) = \begin{bmatrix} 0.1 \\ -1.1 \\ 3.7 \\ 2.5 \end{bmatrix}$$

- At some time t_1, we observe more parts of the function
- We need to update the best approximation
  - How do we find these approximations?
  - How can we effectively update the approximations?



$$x(t_0) = \begin{bmatrix} 0.1 \\ -1.1 \\ 3.7 \\ 2.5 \end{bmatrix} \xrightarrow{\text{update}} x(t_1) = \begin{bmatrix} 1.5 \\ 2.9 \\ -0.3 \\ 2.0 \end{bmatrix}$$

- Formally, given a continuous function $u(t)$, can we maintain a fixed-size representation $c(t) \in \mathbb{R}^N$ at all times t such that $c(t)$ optimally captures the history of u from times 0 to t ?
- Quality of approximation
  - We need to specify a measure (or weight function) that tells us how much we care about every time in the past.
  - For example, EMA uses an exponential decaying measure.
- Basis
  - To represent a function in a fixed-size vector, We can project the function onto a subspace of dimension N and store the N coefficients of its expansion in any basis.
  - For simplicity, we will assume that we are working with the polynomial basis.

- It can be shown that given a measure (and assuming the polynomial basis), the online function approximation problem is fully specified.
- How can we calculate them?
  - See full derivation in the paper. (They used classical tools in optimization theory, such as orthogonal polynomials)
- The solution takes on the form of a simple linear differential equation (High-order Polynomial Projection Operator)

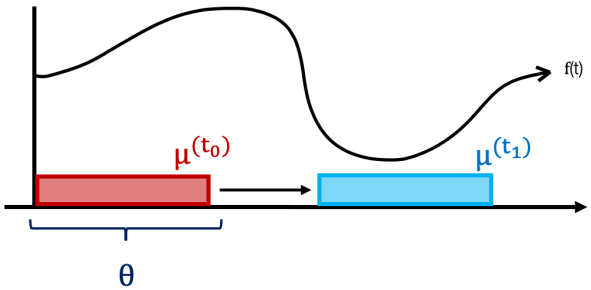
$$\dot{c}(t) = A(t)c(t) + B(t)u(t)$$

- In short, the HiPPO framework takes a **family of measures**, and gives an ODE with **closed-form transition matrices** $A(t), B(t)$ that depend on the measure. Following these dynamics, $c(t)$ optimally approximate the history of $u(t)$.

- Two examples of HiPPO operator
- LegT - fixed-length sliding window, recent past
- LegS - uniformly see the entire history up to the current time

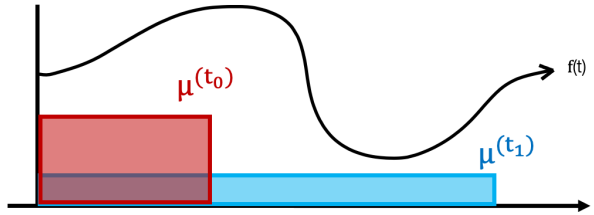
Translated Legendre Measure (LegT)

$$\mu^{(t)} = \frac{1}{\theta}\mathbb{I}[t-\theta, t]$$



Scaled Legendre Measure (LegS)

$$\mu^{(t)} = \frac{1}{t}\mathbb{I}[0, t]$$



HiPPO-LegT operator $\frac{d}{dt}c(t) = -Ac(t) + Bf(t)$

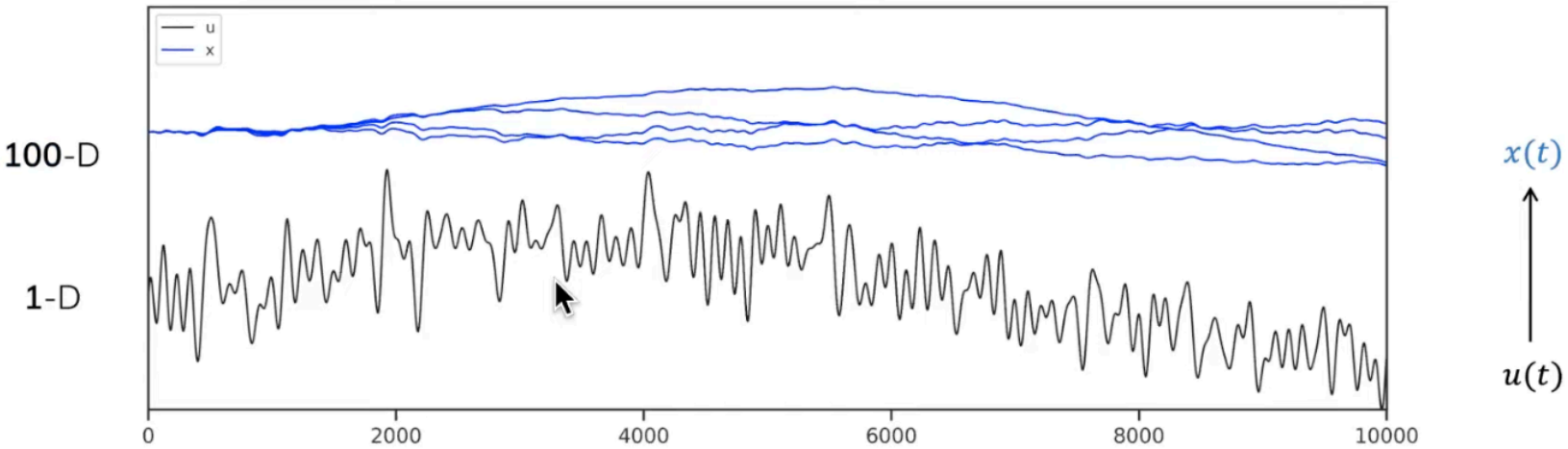
$$A_{nk} = \frac{1}{\theta}\begin{cases}(-1)^{n-k}(2n+1) & \text{if } n \geq k \\ 2n+1 & \text{if } n \leq k\end{cases}, \quad B_n = \frac{1}{\theta}(2n+1)(-1)^n$$

HiPPO-LegS operator $\frac{d}{dt}c(t) = -\frac{1}{t}Ac(t) + \frac{1}{t}Bf(t)$

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

- HiPPO: map 1D to ND
- In DNN, the dimension would blow up

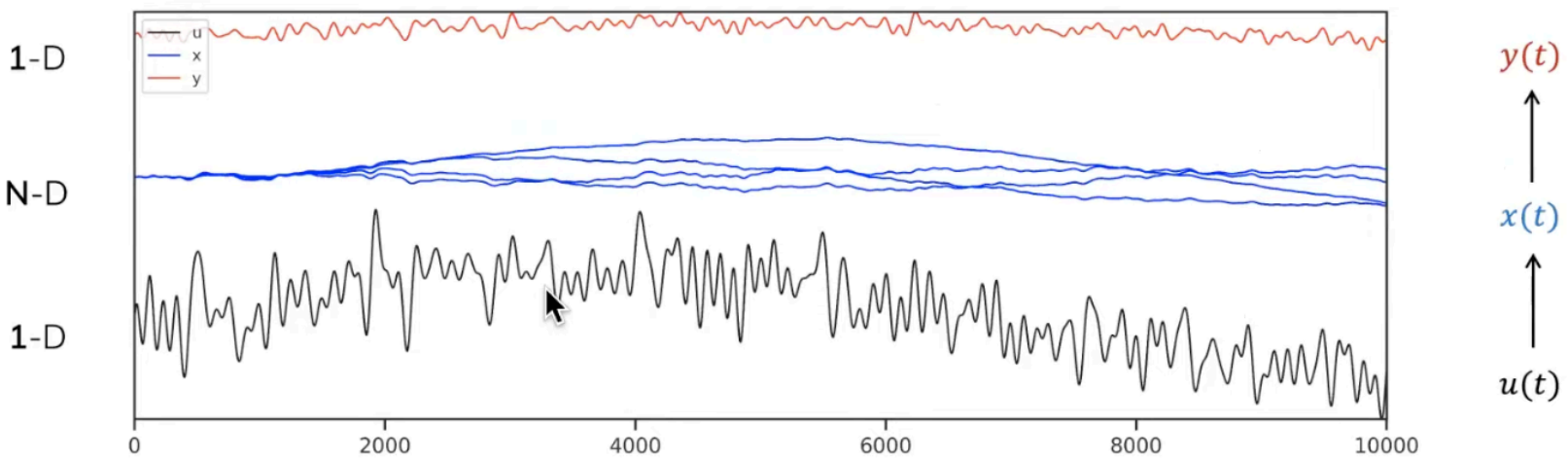
$$x' = \boldsymbol{A}x + \boldsymbol{B}u$$



⚠ Blows up dimension of input in DNN

- We can just decrease the dimension again
- Just use a linear combination C and a skip connection D

$$y = \boldsymbol{C}x + \boldsymbol{D}u$$



S4 projects the **state** to an **output**

- In control engineering, this is called State Space Model (SSM)

Input → State

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

Parameters

$\boldsymbol{A} \in \mathbb{R}^{N \times N}$

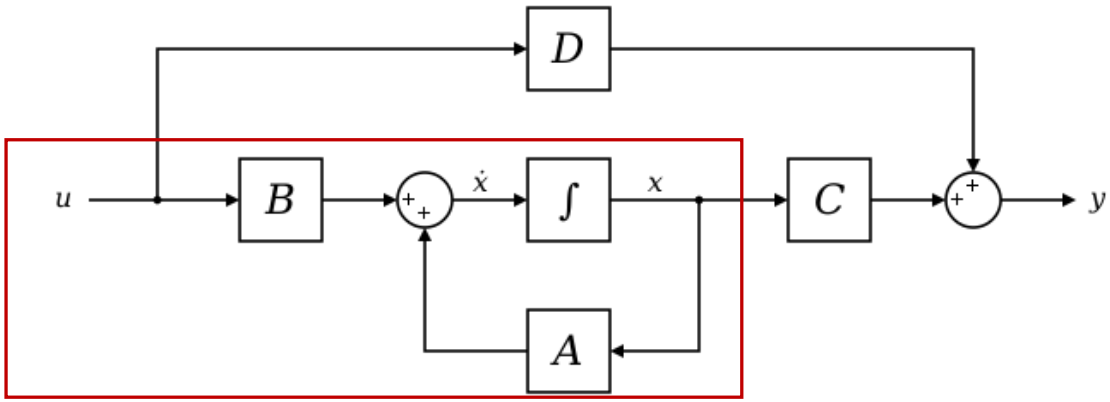
$\boldsymbol{B} \in \mathbb{R}^{N \times 1}$

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

$\boldsymbol{D} \in \mathbb{R}^{1 \times 1}$

Function-to-function map $u(t) \mapsto y(t)$

1D input
u(t)

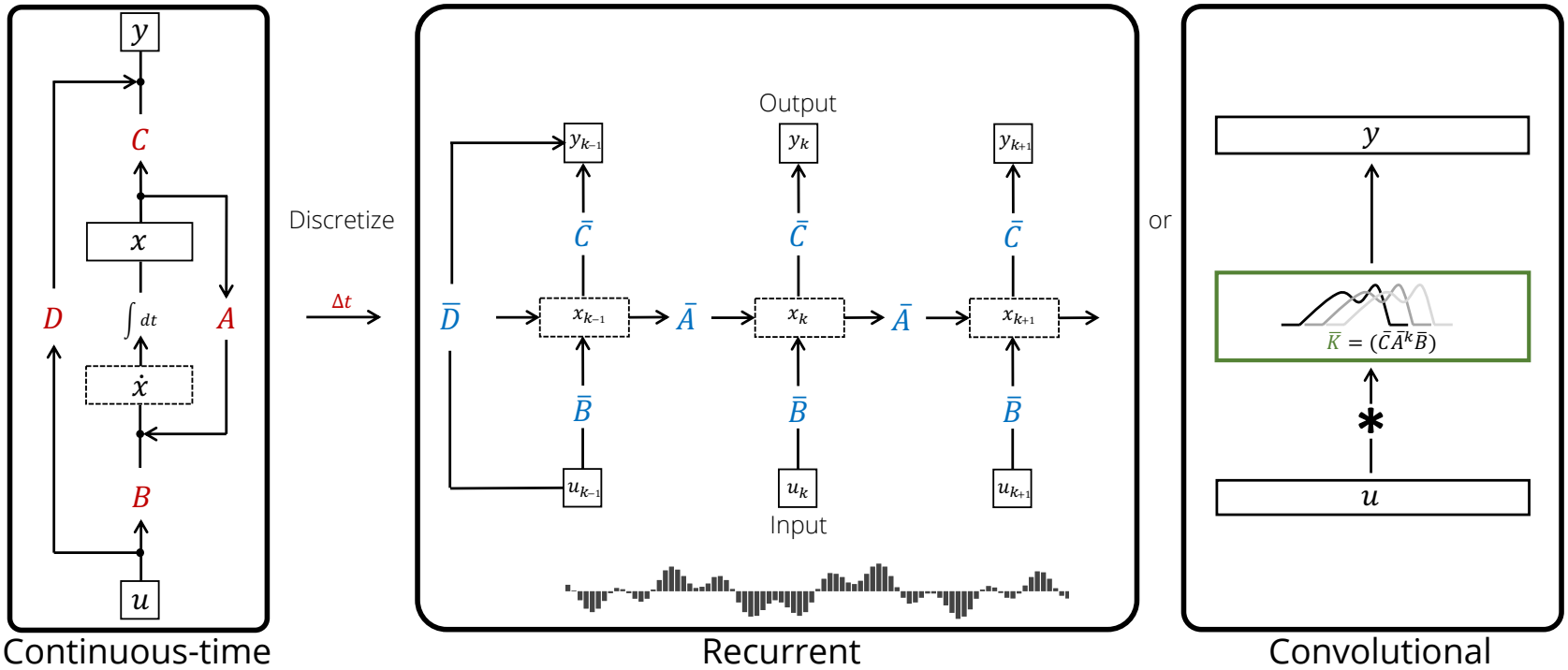


1D output
y(t)

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems.

S4 Model

- This is the Continuous-time view of S4
- Now let's proceed to the Recurrent view



S4 Model

- To apply SSM to a discrete input sequence $(u_0, u_1, \dots)$, it need to be discretized by step size Δ
- The input could be viewed as a sampling of $u(t)$, where $u_k = u(k\Delta)$
- Using the *bilinear method*, which converts the state matrix $\boldsymbol{A}$ into an approximation $\overline{\boldsymbol{A}}$. The discrete SSM is: ($\boldsymbol{D}$ is omitted)

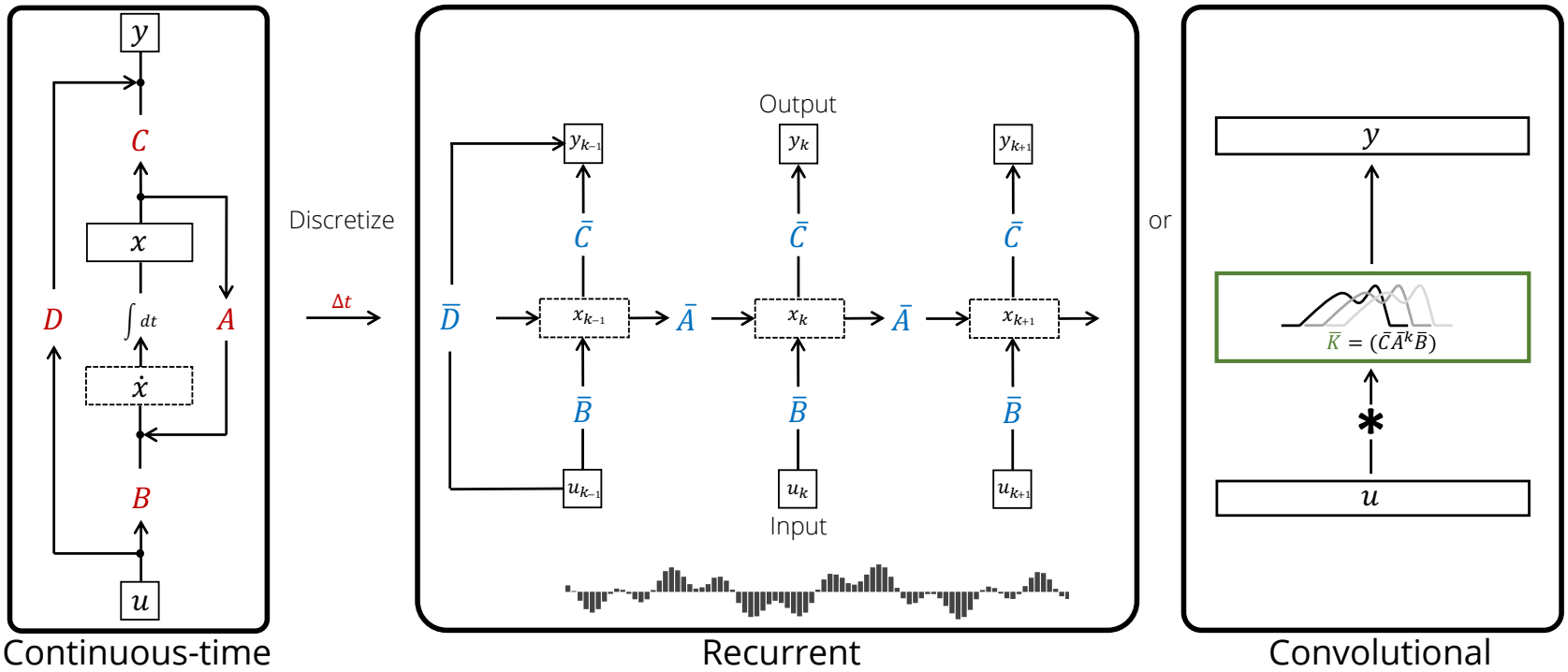
$$\overline{\boldsymbol{A}} = (\boldsymbol{I} - \Delta/2 \cdot \boldsymbol{A})^{-1}(\boldsymbol{I} + \Delta/2 \cdot \boldsymbol{A})$$
$$\overline{\boldsymbol{B}} = (\boldsymbol{I} - \Delta/2 \cdot \boldsymbol{A})^{-1}\Delta \boldsymbol{B}$$
$$\overline{\boldsymbol{C}} = \boldsymbol{C}$$

- Now, it turned into a sequence-to-sequence map $u_k \mapsto y_k$
- It is also a recurrence in x_k (RNN). $x_k \in \mathbb{R}^N$ can be viewed as a hidden state with transition matrix $\overline{\boldsymbol{A}}$:

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

S4 Model

- How to turn the RNN into a Convolution?



S4 Model

- For simplicity let the initial state be $x_{-1} = 0$. Then unrolling explicitly yields:

$$x_0 = \overline{\boldsymbol{B}}u_0 \quad x_1 = \overline{\boldsymbol{AB}}u_0 + \overline{\boldsymbol{B}}u_1 \quad x_2 = \overline{\boldsymbol{A}}^2\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{AB}}u_1 + \overline{\boldsymbol{B}}u_2$$

$$y_0 = \overline{\boldsymbol{CB}}u_0 \quad y_1 = \overline{\boldsymbol{CAB}}u_0 + \overline{\boldsymbol{CB}}u_1 \quad y_2 = \overline{\boldsymbol{CA}}^2\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{CAB}}u_1 + \overline{\boldsymbol{CB}}u_2$$

- This can be vectorized into a convolution with an explicit formula for the convolution kernel:

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

$$y = \overline{\boldsymbol{K}} * u$$

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

- This kernel is Huge!

S4 Model

- Why don't people use SSM with deep learning?
- SSMs inherit properties of CTMs, RNNs, CNNs... including problems with LRDs
  - With randomized $\boldsymbol{A}$, 50% accuracy in MNIST
  - Solved by HiPPO operator, increased to 99% accuracy
- Computation problems

$$\overline{\boldsymbol{K}} = \left(\overline{\boldsymbol{C}}\overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}, \ldots, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}}\right)$$

  - Powering up $\boldsymbol{A} \to$ vanishing gradients
  - Powering up $\boldsymbol{A} \to O(N^2 L)$ time
- The main contribution of S4 is a quick and stable way to sample $\overline{\boldsymbol{K}}$

S4 Model

- S4 tries to speed up the generation of kernel under the Diagonal Plus Low-Rank (DPLR) assumption
  - A DPLR SSM is $(\boldsymbol{\Lambda} - \boldsymbol{P}\boldsymbol{Q}^*, \boldsymbol{B}, \boldsymbol{C})$ for some diagonal $\boldsymbol{\Lambda}$ and matrices $\boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{C}^{N \times 1}$.
- Under the DPLR assumption, $\overline{\boldsymbol{K}}$ is calculated in 3 steps
  1. Instead of computing $\overline{\boldsymbol{K}}$ directly, we compute its spectrum by evaluating its *truncated generating function*. This now involves a matrix inverse instead of power.
  2. We show that the diagonal matrix case is equivalent to the computation of a *Cauchy kernel* $\frac{1}{\omega_j - \zeta_k}$.
  3. We show the low-rank term can now be corrected by applying the *Woodbury identity* which reduces $(\boldsymbol{\Lambda} + \boldsymbol{P}\boldsymbol{Q}^*)^{-1}$ in terms of $\boldsymbol{\Lambda}^{-1}$, truly reducing to the diagonal case.

- The truncated SSM generating function at z with truncation L is

$$\hat{\mathcal{K}}_L(z; \overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}) \in \mathbb{C} := \sum_{i=0}^{L-1} \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^i \overline{\boldsymbol{B}} z^i$$

- The generating function converts the SSM convolution filter from the time domain to frequency domain (also called *z-transform*)
- By evaluating of the z-transform at the roots of unity $\Omega = \left\{\exp\left(2\pi \frac{k}{L}\right. : k \in [L]\right\}$ and inverse fourier transformation, we can recover the filter.
- The purpose of this operation is to replace the matrix power with an inverse

$$\hat{\mathcal{K}}_L(z) = \sum_{i=0}^{L-1} \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^i \overline{\boldsymbol{B}} z^i = \overline{\boldsymbol{C}}\left(\boldsymbol{I} - \overline{\boldsymbol{A}}^L z^L\right)(\boldsymbol{I} - \overline{\boldsymbol{A}} z)^{-1}\overline{\boldsymbol{B}} = \widetilde{\boldsymbol{C}}(\boldsymbol{I} - \overline{\boldsymbol{A}} z)^{-1}\overline{\boldsymbol{B}}$$

- However this inverse still needs to be calculated L times (for each of the roots of unity).

- In this step, we assume special structures of $\boldsymbol{A}$ to get fast inverse
- First convert back to the original SSM by expanding discretization

$$\widetilde{\boldsymbol{C}}(\boldsymbol{I}-\overline{\boldsymbol{A}})^{-1}\overline{\boldsymbol{B}}=\frac{2\Delta}{1+z}\widetilde{\boldsymbol{C}}\left[2\frac{1-z}{1+z}-\Delta\boldsymbol{A}\right]^{-1}\boldsymbol{B}$$

- Suppose $\boldsymbol{A}=\boldsymbol{\Lambda}$ for a diagonal $\boldsymbol{\Lambda}$. Plug this in and we have,

$$\hat{\boldsymbol{K}}_{\boldsymbol{\Lambda}}(z)=c(z)\sum_i\frac{\widetilde{\boldsymbol{C}}_i\boldsymbol{B}_i}{(g(z)-\boldsymbol{\Lambda}_i)}=c(z)\cdot k_{z,\boldsymbol{\Lambda}}(\widetilde{\boldsymbol{C}},\boldsymbol{B})$$

  where c is a constant, and g is a function of z.
- Here the inverse is replaced with a weighted dot product
- Note this is a *Cauchy kernel* and can be calculated *very fast*

- Finally, we relax the diagonal assumption. In addition to the diagonal term we allow a low-rank component with $\boldsymbol{P}, \boldsymbol{Q} \in \mathbb{C}^{N \times 1}$

$$\boldsymbol{A} = \boldsymbol{\Lambda} - \boldsymbol{P}\boldsymbol{Q}^*$$

- The *Woodbury identity* shows the inverse of a diagonal plus rank-1 term is equal to the inverse of the diagonal plus a rank-1 term:

$$\left(\boldsymbol{\Lambda} + \boldsymbol{P}\boldsymbol{Q}^*\right)^{-1} = \boldsymbol{\Lambda}^{-1} - \boldsymbol{\Lambda}^{-1}\boldsymbol{P}\left(1 + \boldsymbol{Q}^*\boldsymbol{\Lambda}^{-1}\boldsymbol{P}\right)^{-1}\boldsymbol{Q}^*\boldsymbol{\Lambda}^{-1}$$

- After applying the Woodbury identity we get

$$\hat{\boldsymbol{K}}_{DPLR}(z) = c(z)\left[k_{z,\Lambda}(\widetilde{\boldsymbol{C}}, \boldsymbol{B}) - k_{z,\Lambda}(\widetilde{\boldsymbol{C}}, \boldsymbol{P})\left(1 + k_{z,\Lambda}\left(\boldsymbol{q}^*, \boldsymbol{P}\right)\right)^{-1} k_{z,\Lambda}\left(\boldsymbol{q}^*, \boldsymbol{B}\right)\right]$$

- This contains four Cauchy kernels

- $\boldsymbol{A}$ being DPLR has another advantage in the discretized SMM
- The discretization is

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

- We simplify both terms in the definition of $\overline{\boldsymbol{A}}$ independently. The first term is:

$$\begin{aligned}\boldsymbol{I} + \frac{\Delta}{2}\boldsymbol{A} &= \boldsymbol{I} + \frac{\Delta}{2}(\boldsymbol{\Lambda} - \boldsymbol{P}\boldsymbol{Q}^*) \\ &= \frac{\Delta}{2}\left[\frac{2}{\Delta}\boldsymbol{I} + (\boldsymbol{\Lambda} - \boldsymbol{P}\boldsymbol{Q}^*)\right] \\ &= \frac{\Delta}{2}\boldsymbol{A_0}\end{aligned}$$

where $\boldsymbol{A_0}$ is defined as the term in the final brackets.

- The second term is known as the Backward Euler's method, which utilities the Woodbury's Identity

$$\begin{aligned}\left(\boldsymbol{I}-\frac{\Delta}{2}\boldsymbol{A}\right)^{-1} &= \left(\boldsymbol{I}-\frac{\Delta}{2}\left(\boldsymbol{\Lambda}-\boldsymbol{P}\boldsymbol{Q}^{*}\right)\right)^{-1} \\ &= \frac{2}{\Delta}\left[\frac{2}{\Delta}-\boldsymbol{\Lambda}+\boldsymbol{P}\boldsymbol{Q}^{*}\right]^{-1} \\ &= \frac{2}{\Delta}\left[\boldsymbol{D}-\boldsymbol{D}\boldsymbol{P}\left(1+\boldsymbol{Q}^{*}\boldsymbol{D}\boldsymbol{P}\right)^{-1}\boldsymbol{Q}^{*}\boldsymbol{D}\right] \\ &= \frac{2}{\Delta}\boldsymbol{A}_{1}\end{aligned}$$

  where $\boldsymbol{D}=\left(\frac{2}{\Delta}-\boldsymbol{\Lambda}\right)^{-1}$ and $\boldsymbol{A}_1$ is the term in the final brackets.

- The discrete-time SSM becomes

$$\begin{aligned}x_k &= \overline{\boldsymbol{A}}x_{k-1}+\overline{\boldsymbol{B}}u_k \\ &= \boldsymbol{A_1}\boldsymbol{A_0}x_{k-1}+2\boldsymbol{A_1}\boldsymbol{B}u_k \\ y_k &= \boldsymbol{C}x_k\end{aligned}$$

- We want to use the HiPPO operator as $\boldsymbol{A}$, which is not DPLR.
- Fortunately, it is Normal Plus Low-Rank (NPLR).
- The S4 techniques can apply to any matrix $\boldsymbol{A}$ that can be decomposed as Normal Plus Low-Rank (NPLR).

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

  for unitary $\boldsymbol{V} \in \mathbb{C}^{N\times N}$, diagonal $\boldsymbol{\Lambda}$, and low-rank $\boldsymbol{P}, \boldsymbol{Q} \in \mathbb{R}^{N\times r}$.
- The authors further showed that the normal part can be written as a *Skew-symmetric* matrix, which allows us the use the quick diagonalization algorithms for *Hermitian matrices*

- A summary of Kernel Generation

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

Sequential Image Classification

	Model	sMNIST	pMNIST	sCIFAR
Transformers	Transformer (Vaswani et al., 2017; Trinh et al., 2018)	98.9	97.9	62.2
CNNs	CKConv (Romero et al., 2021)	99.32	98.54	63.74
	TrellisNet (Bai et al., 2019)	99.20	98.13	73.42
	TCN (Bai et al., 2018)	99.0	97.2	-
RNNs	LSTM (Hochreiter & Schmidhuber, 1997; Gu et al., 2020b)	98.9	95.11	63.01
	r-LSTM (Trinh et al., 2018)	98.4	95.2	72.2
	Dilated GRU (Chang et al., 2017)	99.0	94.6	-
	Dilated RNN (Chang et al., 2017)	98.0	96.1	-
	IndRNN (Li et al., 2018)	99.0	96.0	-
	expRNN (Lezcano-Casado & Martínez-Rubio, 2019)	98.7	96.6	-
	UR-LSTM	99.28	96.96	71.00
	UR-GRU (Gu et al., 2020b)	99.27	96.51	74.4
	LMU (Voelker et al., 2019)	-	97.15	-
	HiPPO-RNN (Gu et al., 2020a)	98.9	98.3	61.1
	UNIcoRNN (Rusch & Mishra, 2021)	-	98.4	-
	LMUFFT (Chilkuri & Eliasmith, 2021)	-	98.49	-
	LipschitzRNN (Erichson et al., 2021)	99.4	96.3	64.2
SSMs	**S4**	**99.63**	**98.70**	**91.13**

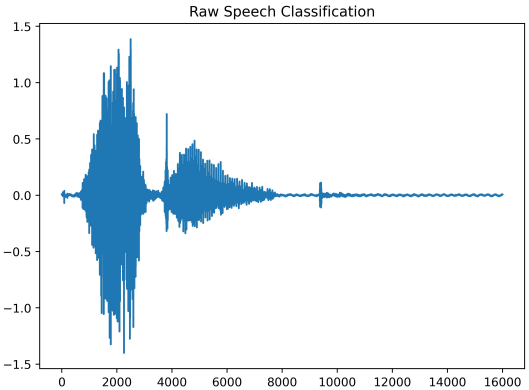
Setting	**S4**	ResNet
-Norm	**90.46**	79.52
Base	**91.12**	89.46
+Aug	93.16	**95.62**

- Speech is difficult because LRD: 1 second clip = length-16000
- Speech pipelines require feature engineering
- e.g. Mel-frequency Cepstrum Coefficients (MFCC) (length 160)

	MFCC	
Transformer	90.75	**Transformers**
Performer	80.85	
ODE-RNN	65.9	**CTMs**
NRDE	89.8	
ExpRNN	82.13	**RNNs**
LipschitzRNN	88.38	
CKConv	**95.3**	**CNNs**
WaveGAN-D	✗	
LSSL	93.58	**SSMs**
S4	93.96	

- Speech Classification

Raw data requires specialized CNNs



	L=160	L=16000
	MFCC	Raw
Transformer	90.75	✗
Performer	80.85	30.77
ODE-RNN	65.9	✗
NRDE	89.8	16.49
ExpRNN	82.13	11.6
LipschitzRNN	88.38	✗
CKConv	**95.3**	71.66
WaveGAN-D	✗	96.25
LSSL	93.58	✗
S4	93.96	**98.32**

← **88x larger than S4**

- Speech Classification

Irregular Continuous Data

Missing values

Varying sampling rates

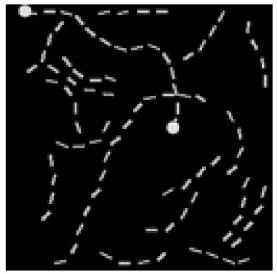
		Train: 16K Hz	**Test: 8K Hz**
	MFCC	RAW	0.5×
Transformer	90.75	✗	✗
Performer	80.85	30.77	30.68
ODE-RNN	65.9	✗	✗
NRDE	89.8	16.49	15.12
ExpRNN	82.13	11.6	10.8
LipschitzRNN	88.38	✗	✗
CKConv	**95.3**	71.66	65.96
WaveGAN-D	✗	96.25	✗
LSSL	93.58	✗	✗
S4	93.96	**98.32**	**96.30**

Long Range Arena

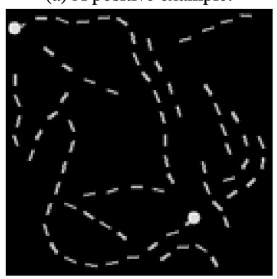
Benchmark spanning text, images, symbolic reasoning (length 1K-16K)

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Random	10.00	50.00	50.00	10.00	50.00	50.00	36.67
Transformer	36.37	64.27	57.46	42.44	71.40	✗	53.66
Local Attention	15.82	52.98	53.39	41.46	66.63	✗	46.71
Sparse Trans.	17.07	63.58	59.59	44.24	71.71	✗	51.03
Longformer	35.63	62.85	56.89	42.22	69.71	✗	52.88
Linformer	35.70	53.94	52.27	38.56	76.34	✗	51.14
Reformer	37.27	56.10	53.40	38.07	68.50	✗	50.56
Sinkhorn Trans.	33.67	61.20	53.83	41.23	67.45	✗	51.23
Synthesizer	36.99	61.68	54.67	41.61	69.45	✗	52.40
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	**58.35**	**76.02**	**87.09**	**87.26**	**86.05**	**88.10**	**80.48**

Path-X



(a) A positive example.



(b) A negative example.

References

[1] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Ré, "Hippo: Recurrent memory with optimal polynomial projections," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1474–1487, 2020.

[2] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," *arXiv preprint arXiv:2111.00396*, 2021.