Efficiently Modeling Long Sequences with Structured State Spaces

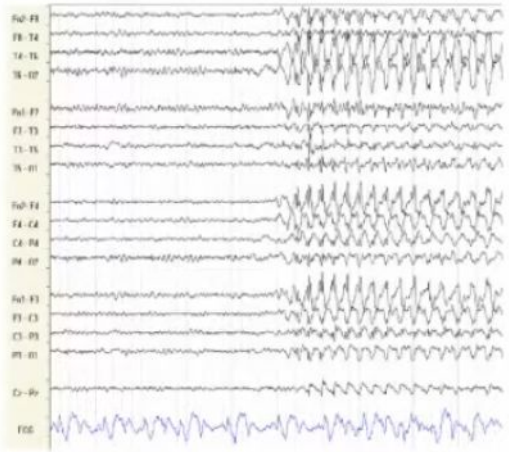
By Albert Gu, Karan Goel, and Christopher Ré

CPSC 670
Spring 2023
David Peng, Yujie Qiao

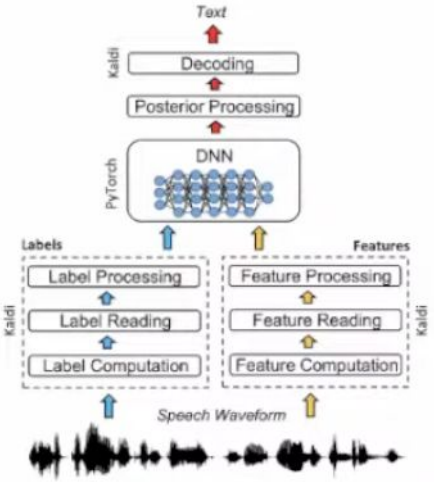
Outline

- Introduction
- Background
- Methods
- Experiments
- Conclusions
- Discussion

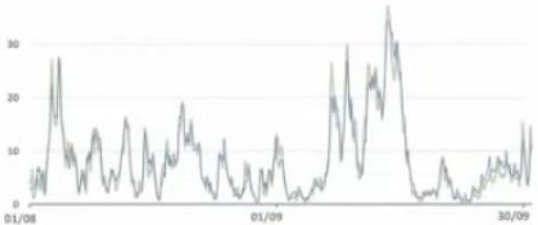
Data/task: Long “Continuous” Time Series



EEG/ECG



Speech



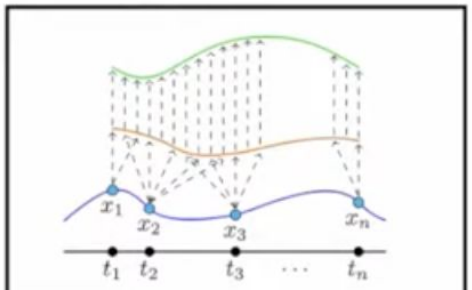
Energy Forecasting

Data/task: Long “Continuous” Time Series

Characteristics

- Sequence data that’s sampled from underlying continuous signal
- Tens of thousands of samples
  - e.g. raw recordings of speech are 16,000 Hertz
  - 1-2 second recording → input sequence with tens of thousands of samples
- Often requires feature engineering

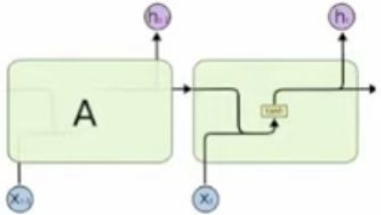
Paradigms for Long Time Series



✓ Continuous data
Irregular sampling

✗ Complex, very inefficient
Vanishing gradients

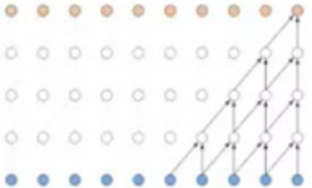
Continuous-time (CTM)



✓ Unbounded context
Stateful inference

✗ Inefficient training
Vanishing gradients

Recurrent (RNN)



✓ Easy optimization
Parallelizable training

✗ Inefficient inference
Bounded context

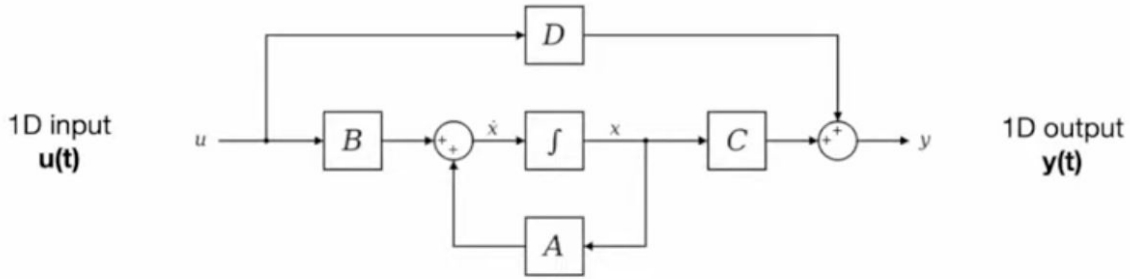
Convolutional (CNN)

State space models (SSMs) can unify the strengths of CTM, RNN, CNN models at handling long-range dependencies

Background: State Space Model (SSM)

- Linear time-invariant system
- Ordinary differential equation (ODE) used commonly in electrical engineering
- Given input signal (sequence) ***u(t)***, produce output signal (sequence) ***y(t)***
- A larger dimensional hidden state ***x(t)*** retains complex patterns
  - drop-in replacement for self-attention

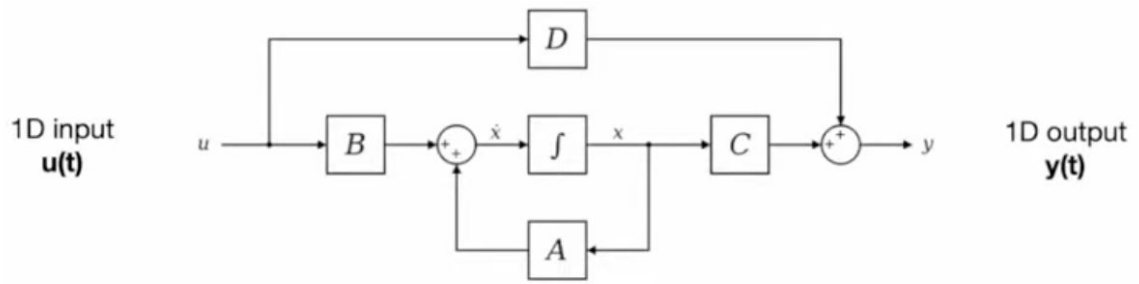
$$x'(t) = \boldsymbol{A}x(t) + \boldsymbol{B}u(t)$$
$$y(t) = \boldsymbol{C}x(t) + \boldsymbol{D}u(t)$$



Background: State Space Model (SSM)

- **A, B, C, D** are matrices
- Treat as parameters to be learned in a deep learning model

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



$$u(t) \in \mathbb{R}^p \quad A \in \mathbb{R}^{d \times d}$$
$$x(t) \in \mathbb{R}^d \quad B \in \mathbb{R}^{d \times p}$$
$$y(t) \in \mathbb{R}^p \quad C \in \mathbb{R}^{1 \times d}$$
$$D \in \mathbb{R}^{d \times 1}$$

Solving the ODEs, we can derive…

$$x(s) = e^{As}x(0) + \boxed{\int_0^s e^{A(s-t)}Bu(t)\,dt}$$

- Given input data and a value of ***A*****,** we could numerically integrate to get the hidden state value at any time ***s***
- Hidden state is a convolution over the input data
- Next step: turn integral into discrete approximation

Turn continuous input into discrete input

- Use step size **Δ**
- ***u[k] = u(kΔ)*** for ***k = 1, 2, …, n***
- Convert state matrices into discrete approximations based on
  - step size Δ
  - state matrix ***A***
- Recurrence relation–like an RNN!

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

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

- ***D*** is set to identity matrix (acts like skip connection)

Refine recurrence relation as convolution

- CNNs are easier to train than RNNs
- Convolution is easier to train than recurrence relation
- Calculate $\overline{\boldsymbol{K}}$, the SSM convolution kernel or filter

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

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

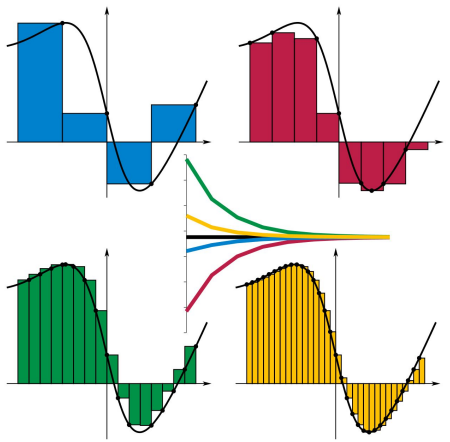
$$y_k = \overline{\boldsymbol{CA}}^k \overline{\boldsymbol{B}} u_0 + \overline{\boldsymbol{CA}}^{k-1} \overline{\boldsymbol{B}} u_1 + \cdots + \overline{\boldsymbol{CAB}} u_{k-1} + \overline{\boldsymbol{CB}} u_k$$
$$y = \overline{\boldsymbol{K}} * u.$$

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

Turning continuous function into discrete function

- Input of the model is actually discrete inputs
- Sample frequency ***T***
- ***u[k] = u(kT)*** for ***k = 1, 2, …, n***
- How can we find ***x[k] = x(kT)*** ?
- We can derive…

$$\int_0^{(n+1)T} f(u(t))dt \approx T\sum_{k=0}^{n} f(u[k])$$



$$x[n+1] = e^{AT}x[n] + Te^{AT}Bu[n]$$

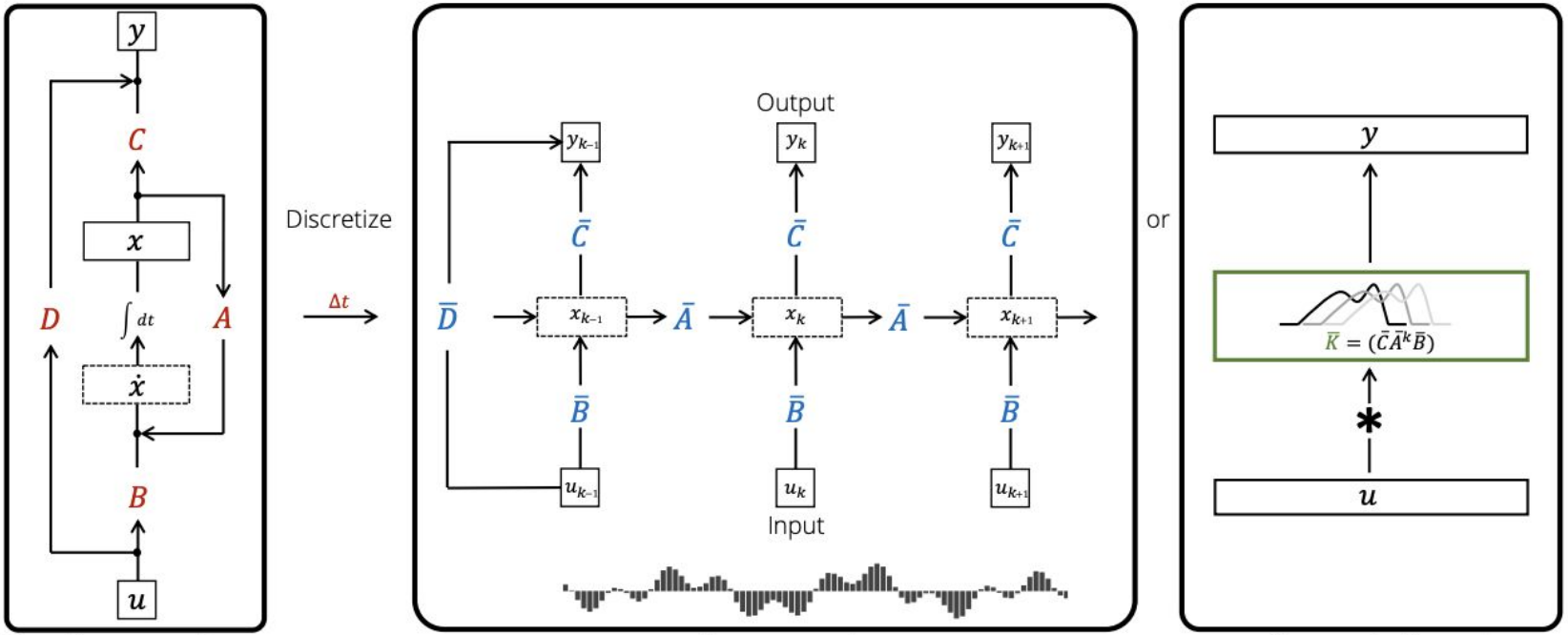
Redefine recurrence relation as convolution

$$x[n+1] = e^{AT}x[n] + Te^{AT}Bu[n]$$

- Convolution is easier to calculate than recurrence
- We can derive…

$$x[n+1] = e^{AT(n+1)}x[0] + T\sum_{j=0}^{k} e^{A(k-j)T}Bu[j]$$

Three forms of SSM



Continuous-time
- ✓ *continuous data*
- ✓ *irregular sampling*

Recurrent
- ✓ *unbounded context*
- ✓ *efficient inference*

Convolutional
- ✓ *local information*
- ✓ *parallelizable training*

Applying SSM to deep learning

- SSM layer acts as a drop-in replacement for attention
- Model learns matrices ***A, B, C, D***
- We want the training model to be
  - Stable: training leads to reasonable output
  - Efficient: minimize number of computations
  - Expressive: can learn non-linear, complex functions

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

Previous work with SSM: Linear State Space Layer (LSSL)

- Paper published a year earlier by Gu et al.
- Proof of concept for SSM at handling long-range dependencies
- Problem: computation is slow and memory inefficient
- *O(N^2 L)* operations for hidden state dimension *N*, sequence length *L*
  - Much more than comparably-sized RNN, CNN

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

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

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

- How can we calculate $\overline{A}^L$ efficiently? Bottleneck problem

LSSL: non-random initialization of matrix ***A***

- Random initialization leads to poor performance
- Use HiPPO Matrix
  - Paper published earlier by Gu et al.
  - Allows ***x(t)*** to memorize history of input ***u(t)***
  - “compresses” past history to be able to reconstruct most of it
  - Only needs to be calculated once

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

New work: Structured State Space sequence model (S4)

- Further exploits HiPPO matrices to efficiently compute $\overline{\boldsymbol{K}}$
- Main contribution: some matrix math tricks…
  - Argue ***A*** is a type of matrix called a **Normal Plus Low-Rank (NPLR)**
  - Rewrite $\overline{\boldsymbol{K}}$ in the form of its truncated generating function
  - Generating function uses matrix inverses instead of matrix powers. Apply an identity and reduce equation to a simpler case
  - Show that reduced case is equivalent to **Cauchy matrices**, which have been shown to be fast and stable

Theorem 3 (S4 Convolution). *Given any step size Δ, computing the SSM convolution filter $\overline{\boldsymbol{K}}$ can be reduced to 4 Cauchy multiplies, requiring only $\widetilde{O}(N+L)$ operations and $O(N+L)$ space.*

We reiterate that Theorem 3 is our core technical contribution

Experiments - Efficiency Benchmarks

S4 is theoretically much more efficient than the LSSL

- 30x faster, 400x less memory usage

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×

Experiments - Long Range Arena (LRA) and raw speech classification

LRA

- Contains 6 tasks with lengths 1K-16K steps
- S4 solves the Path-X task that involves reasoning about LRDs over sequences of length 128 × 128 = 163840

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**

Experiments - Long Range Arena (LRA) and raw speech classification

Raw Speech Classification

- Used the SC10 subset of the Speech Commands dataset
- Classified raw speech (length-16000)
- S4 achieves 98.3% accuracy, higher than all baselines that use the 100× shorter MFCC features

Experiments - First-ever non-random performance on Path-X task

- Pathfinder task but with longer sequence lengths
- 88% accuracy (50% random guessing for all prior models)

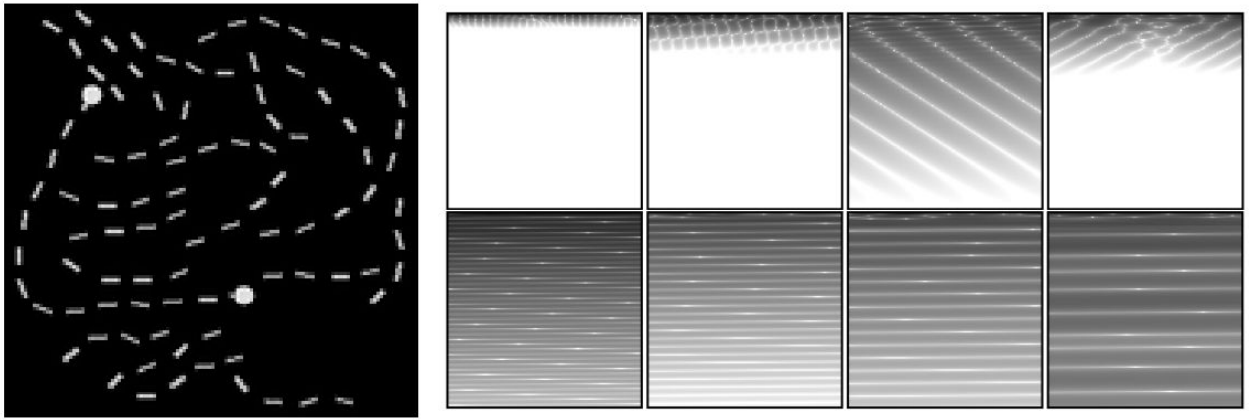


Figure 2: Visualizations of a trained S4 model on LRA Path-X. SSM convolution kernels $\overline{\boldsymbol{K}} \in \mathbb{R}^{16384}$ are reshaped into a 128×128 image. (*Left*) Example from the Path-X task, which involves deducing if the markers are connected by a path (*Top*) Filters from the first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

Experiments - General Sequence Model

Key goal of sequence modeling research is to develop a single model that can be applied in many domains (e.g. images, audio, text, time-series)

S4 can be used as a general sequence model to perform effectively and efficiently in a wide variety of settings including image classification, image and text generation, and time series forecasting

Table 5: (**SC10 classification**) Transformer, CTM, RNN, CNN, and SSM models. (*MFCC*) Standard preprocessed MFCC features (length 161). (*Raw*) Unprocessed signals (length 16000). ($0.5\times$) 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**

Experiments - General Sequence Model

Table 7: (**CIFAR-10 density estimation**) As a generic sequence model, S4 is competitive with previous autoregressive models (in bits per dim.) while incorporating no 2D inductive bias, and has fast generation through its recurrence mode.

Model	bpd	2D bias	Images / sec
Transformer	3.47	**None**	0.32 (1×)
Linear Transf.	3.40	**None**	17.85 (56×)
PixelCNN	3.14	2D conv.	-
Row PixelRNN	3.00	2D BiLSTM	-
PixelCNN++	2.92	2D conv.	19.19 (59.97×)
Image Transf.	2.90	2D local attn.	0.54 (1.7×)
PixelSNAIL	2.85	2D conv. + attn.	0.13 (0.4×)
Sparse Transf.	**2.80**	2D sparse attn.	-
S4 (base)	2.92	**None**	**20.84** (**65.1×**)
S4 (large)	2.85	**None**	3.36 (10.5×)

Table 8: (**WikiText-103 language modeling**) S4 approaches the performance of Transformers with much faster generation. (*Top*) Transformer baseline which our implementation is based on, with attention replaced by S4. (*Bottom*) Attention-free models (RNNs and CNNs).

Model	Params	Test ppl.	Tokens / sec
Transformer	247M	**20.51**	0.8K (1×)
GLU CNN	229M	37.2	-
AWD-QRNN	151M	33.0	-
LSTM + Hebb.	-	29.2	-
TrellisNet	180M	29.19	-
Dynamic Conv.	255M	25.0	-
TaLK Conv.	240M	23.3	-
S4	249M	**20.95**	**48K** (**60×**)

SSM Ablations: the Importance of HiPPO

The HiPPO matrices were critical to initialize an SSM for S4

Investigation on generic SSMs with various initializations

A random Gaussian initialization (with variance scaled down until it did not NaN) & the HiPPO initialization

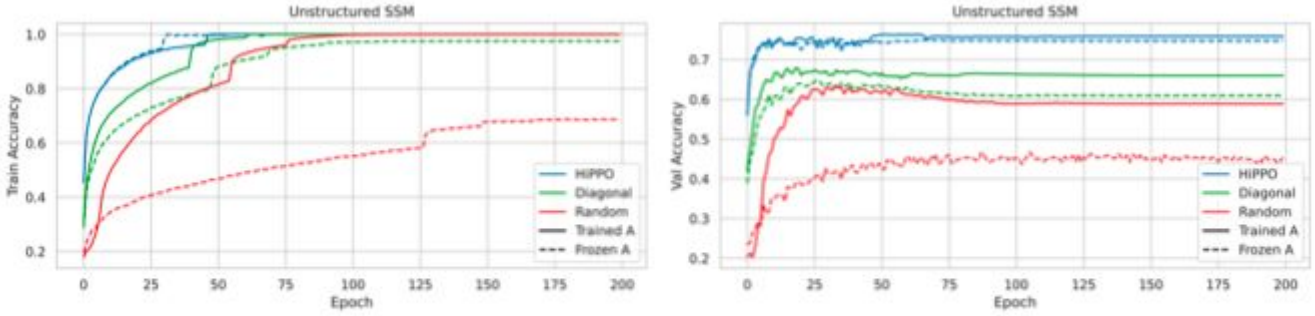


Figure 3: CIFAR-10 classification with unconstrained, real-valued SSMs with various initializations. (*Left*) Train accuracy. (*Right*) Validation accuracy.

SSM Ablations: the Importance of HiPPO

S4's effectiveness primarily comes from the HiPPO initialization, not the NPLR parameterization

The HiPPO initialization (the full S4 method) achieves 84.27% test accuracy with just 100K parameters

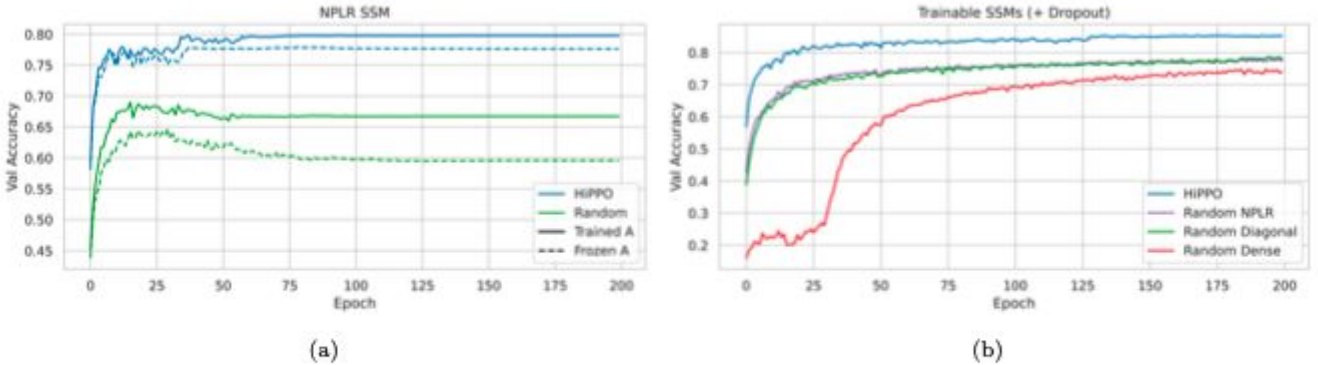


Figure 4: CIFAR-10 validation accuracy of SSMs with different initializations and parameterizations. (*Left*) NPLR parameterization with random versus HiPPO initialization. (*Right*) All methods considered in this section, including minor Dropout regularization. S4 achieves SotA accuracy on sequential CIFAR-10 with just 100K parameters.

Conclusion

Structured State Space sequence model (S4)

- A sequence model that uses a new parameterization for the SSM’s continuous- time, recurrent, and convolutional views to efficiently model LRDs in a principled manner
- S4 achieves strong empirical results across a diverse range of established benchmarks, which suggest that S4 has the potential to be an effective general sequence modeling solution
  - 91% accuracy on sequential CIFAR-10 with no data augmentation or auxiliary losses
  - Comparable to Transformers on image and language modeling tasks, while performing generation 60× faster
  - SoTA on every task from the Long Range Arena benchmark

Discussion

- What are some limitations of S4?
- Does the HiPPO matrices need to be stable?
- How do you think S4 will perform in traditional NLP classification tasks?
- Is there any other ways for memorizing the sequences other than the state space model (SSM)? How would one extend the work in this paper to other directions?