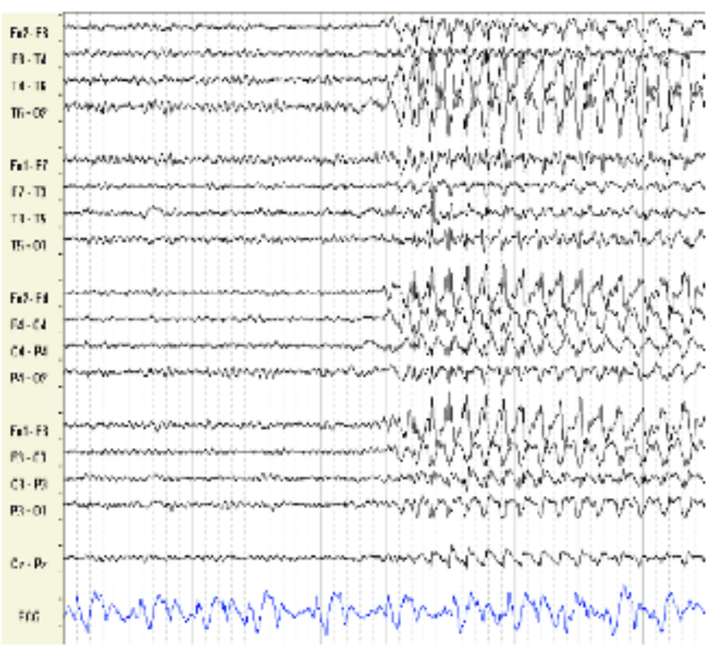
Efficiently Modeling Long Sequences with Structured State Spaces

Advanced Machine Learning in Big Data Analytics

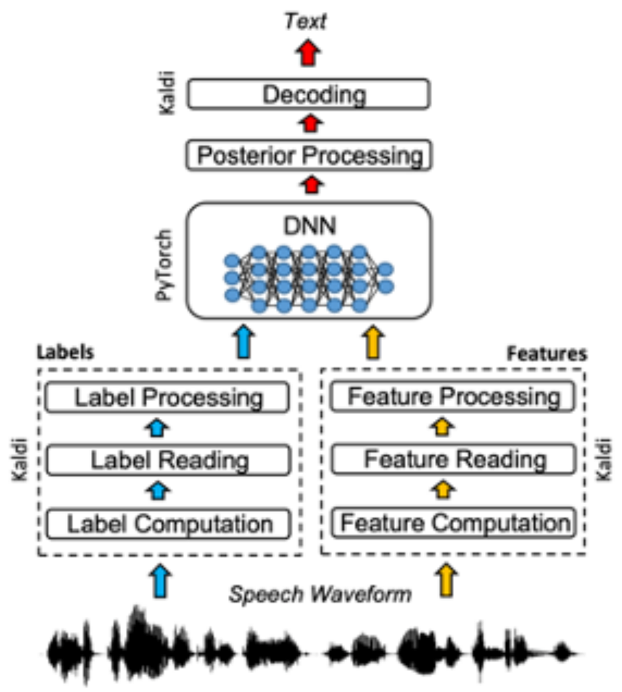
By Maya Natascha Vienken

Table of Contents

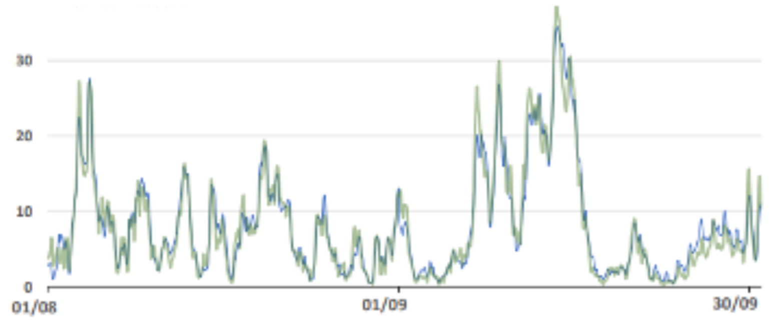
1. Introduction & Problem
2. Long Time Series
   (RNN, CNN, CTM, Transformers)
3. State Space Models (SSMs)
4. Structured State Spaces (**S4**)
5. Their Experiments and Results
6. Further Applications/Conclusion



EEG/ECG



Audio

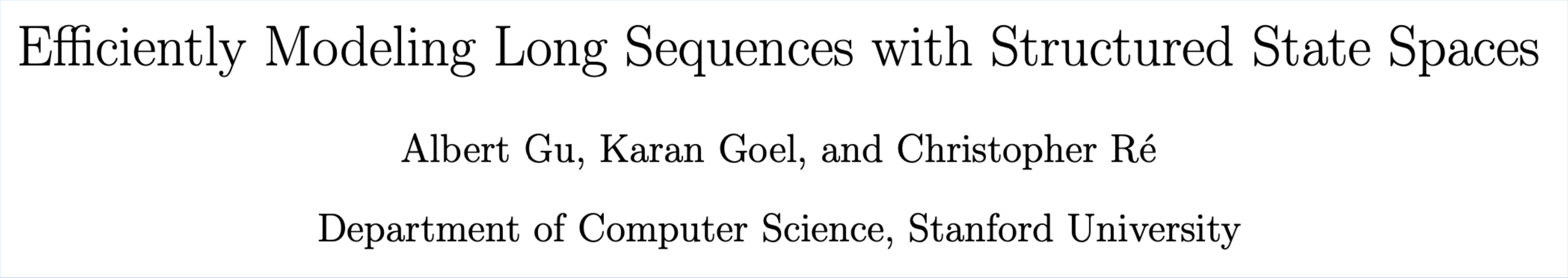


Energy Forecasting

Information on Paper

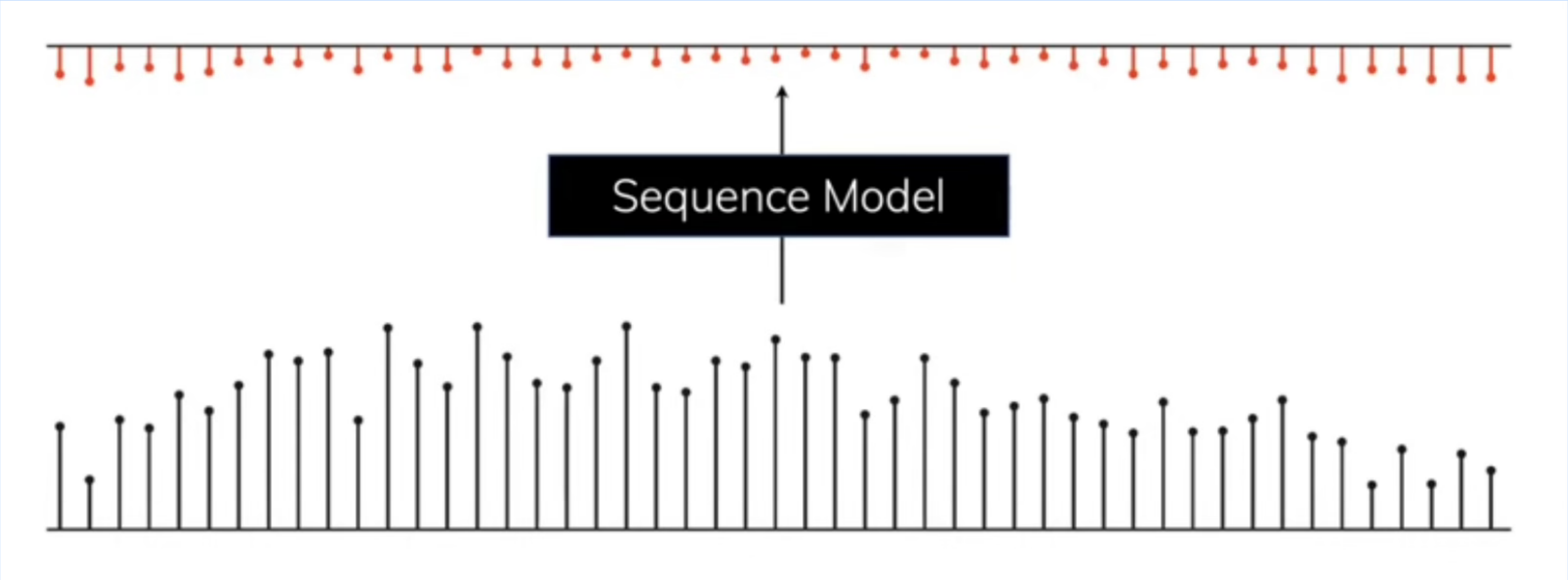
Leveraging S4 for Superior Sequence Modeling

- Published 5th Aug 2022
- Albert Gu: Stanford PhD Student
- Cited by: 926
- Introducing a new sequence model


Efficiently Modeling Long Sequences with Structured State Spaces

Albert Gu, Karan Goel, and Christopher Ré

Department of Computer Science, Stanford University

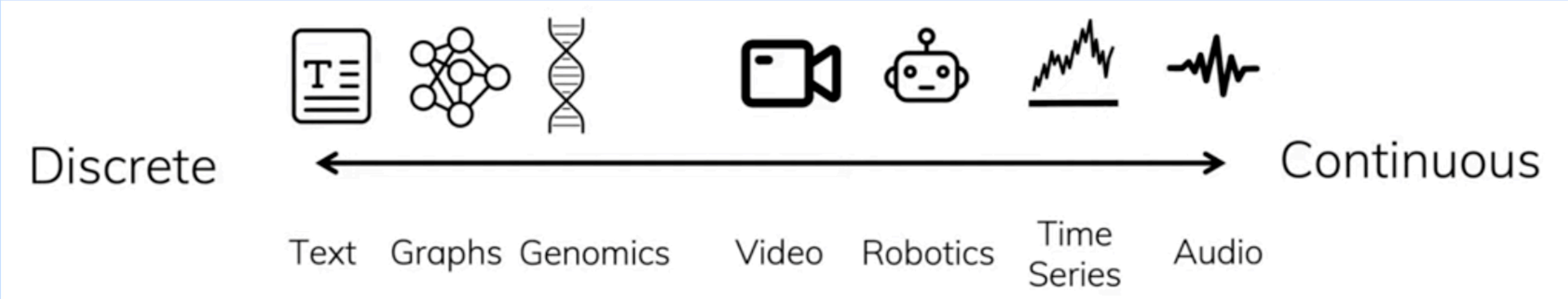


Picture 1: https://arxiv.org/pdf/2111.00396
Picture 2: https://www.youtube.com/watch?v=luCBXCErkCs&si=KLTFClt59BYWsd41

Introduction & Motivation

Sequence Models Struggle with LRDs (Long Range Dependencies)

- Why it matters: Real-world time-series data often **tens of thousands** of time steps
- Types of problems: long and implicitly ***continuous*** sequences
- Goal: designing a single principle model that can address sequence data across a **range of modalities and tasks**, particularly on **LRDs** with **minimal computational resources** (efficient training, fast generation, handling irregularly sampled data)



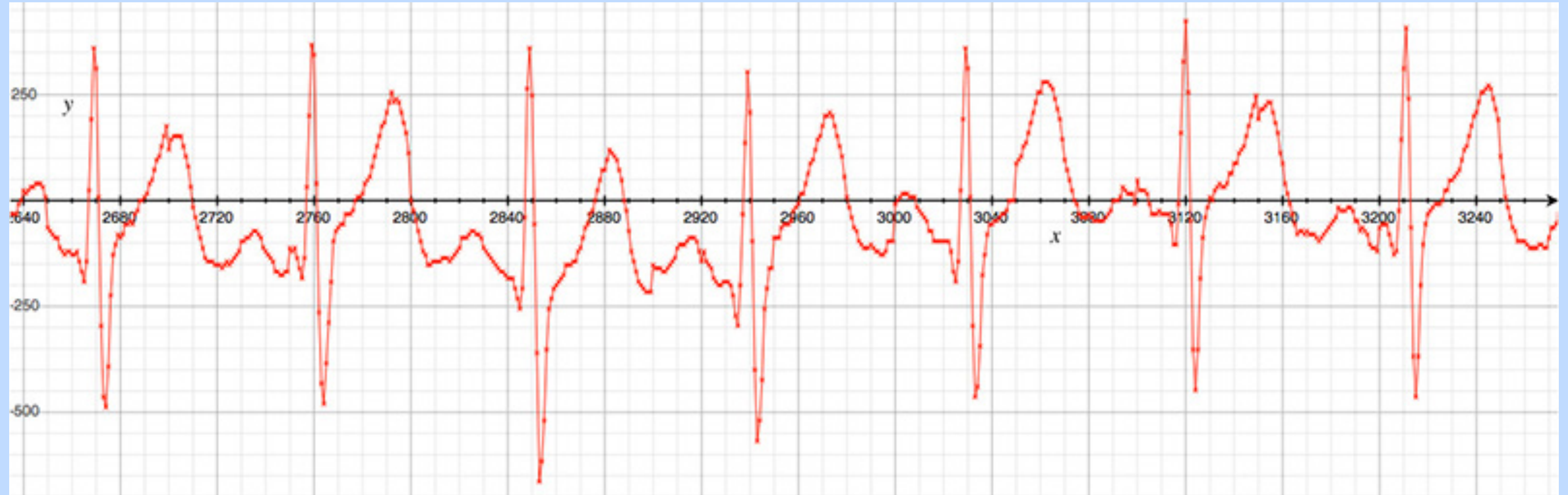
Picture: https://www.youtube.com/live/EvQ3ncuriCM?si=trMHWznpHIDOHOeV

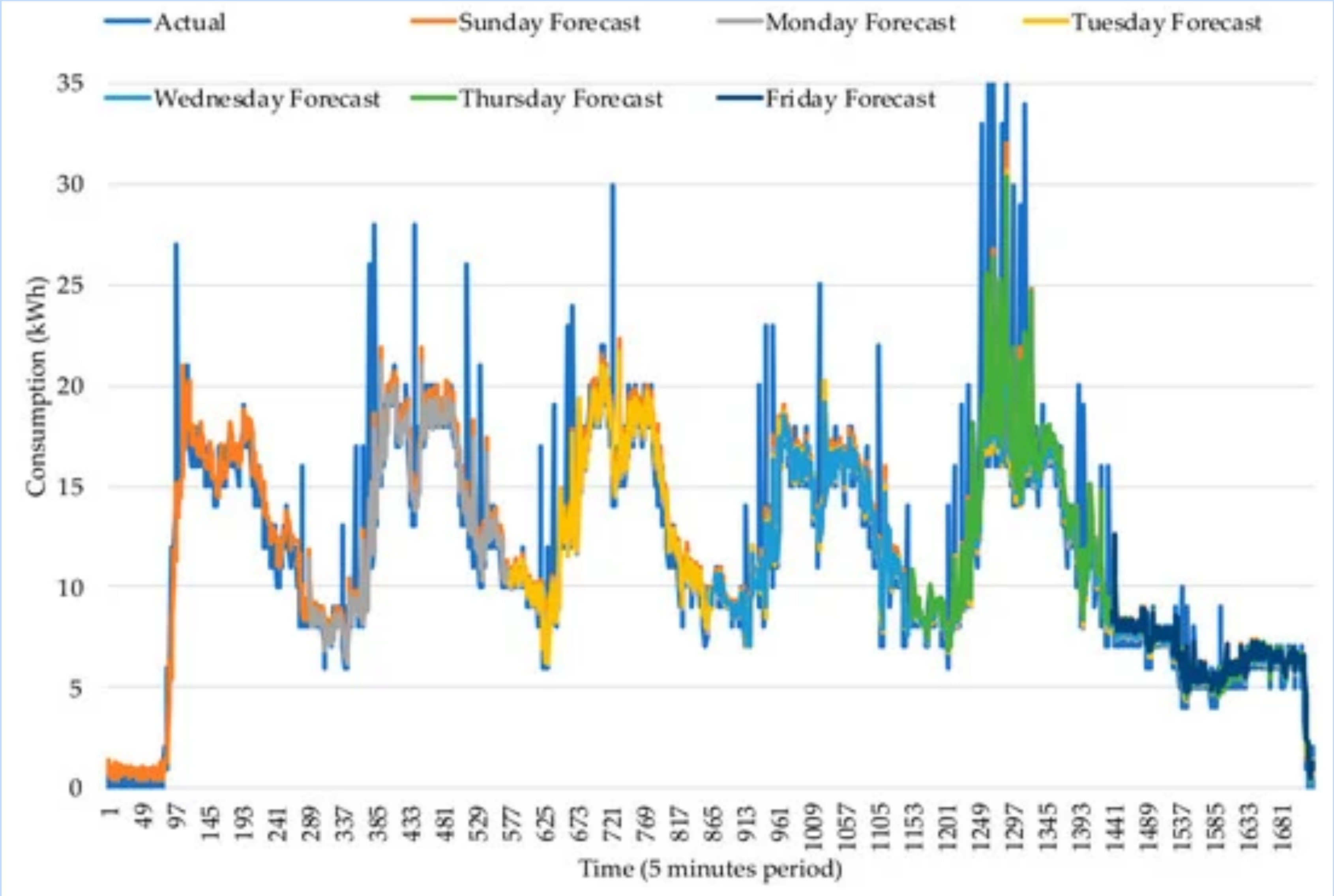
Introduction & Motivation

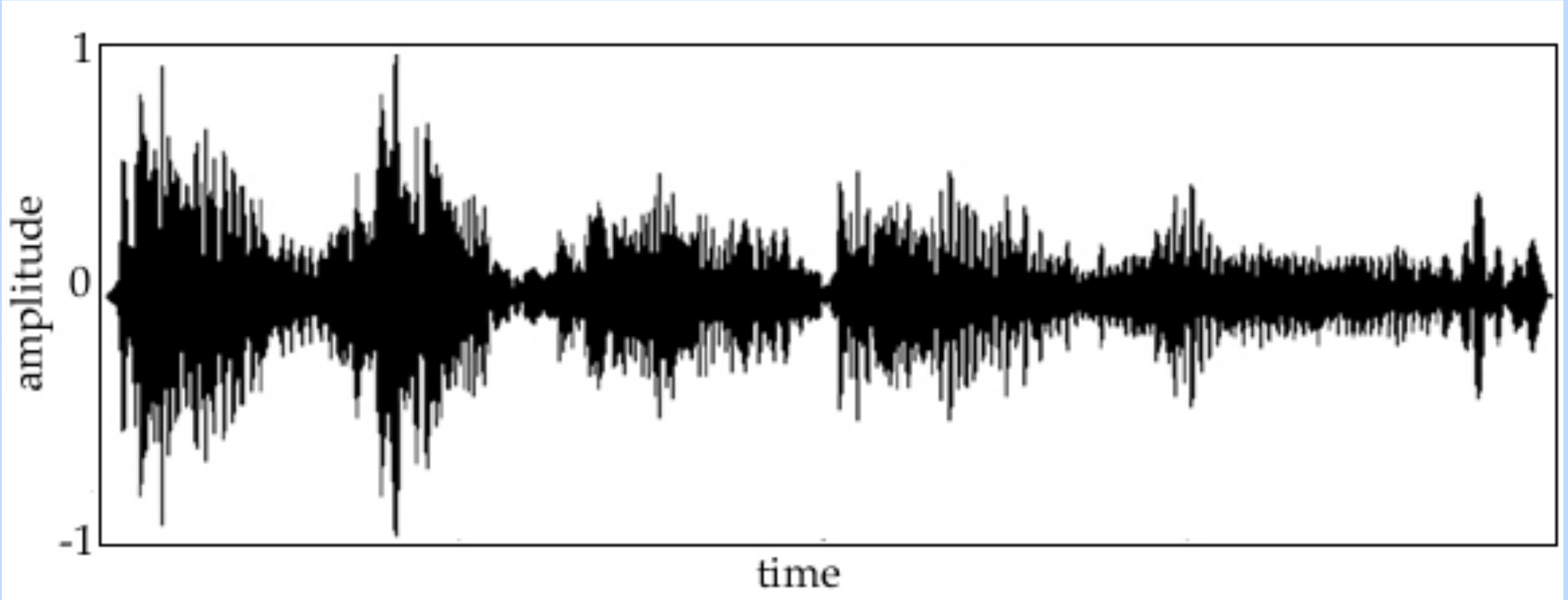
Example Data

- Type of data: Signal data (roughly more continuous data)
  - Time series, Video, Audio, …
  - sampled at high frequency

- Medical time series (EEG/ECG)
- Energy forecasting signals
- Speech waveform
  - Audio waveforms have 16000+ samples per second







Picture 1: https://paulbourke.net/dataformats/holter/channel1.jpg
Picture 2: https://encrypted-tbn2.gstatic.com/images?q=tbn:ANd9GcTwjDGKJ35zSCe5L0Avtg7cqeyFdEd2NSajXB3tHeQcaXzQbAp7
Picture 3: https://musicandcomputersbook.com/images/chapter1/elmowave.jpg

Long Range Arena Benchmark (Classification)

Motivation for some New Model

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Chance	10.00	50.00	50.00	10.00	50.00	50.00	44.00
Transformer	36.37	64.27	57.46	42.44	71.40	FAIL	<u>54.39</u>
Local Attention	15.82	52.98	53.39	41.46	66.63	FAIL	46.06
Sparse Trans.	17.07	63.58	**59.59**	**44.24**	71.71	FAIL	51.24
Longformer	35.63	62.85	56.89	42.22	69.71	FAIL	53.46
Linformer	35.70	53.94	52.27	38.56	<u>76.34</u>	FAIL	51.36
Reformer	**37.27**	56.10	53.40	38.07	68.50	FAIL	50.67
Sinkhorn Trans.	33.67	61.20	53.83	41.23	67.45	FAIL	51.39
Synthesizer	<u>36.99</u>	61.68	54.67	41.61	69.45	FAIL	52.88
BigBird	36.05	64.02	<u>59.29</u>	40.83	74.87	FAIL	**55.01**
Linear Trans.	16.13	**65.90**	53.09	42.34	75.30	FAIL	50.55
Performer	18.01	<u>65.40</u>	53.82	<u>42.77</u>	**77.05**	FAIL	51.41
Task Avg (Std)	29 (9.7)	61 (4.6)	55 (2.6)	41 (1.8)	72 (3.7)	FAIL	52 (2.4)

Picture: https://arxiv.org/pdf/2111.00396

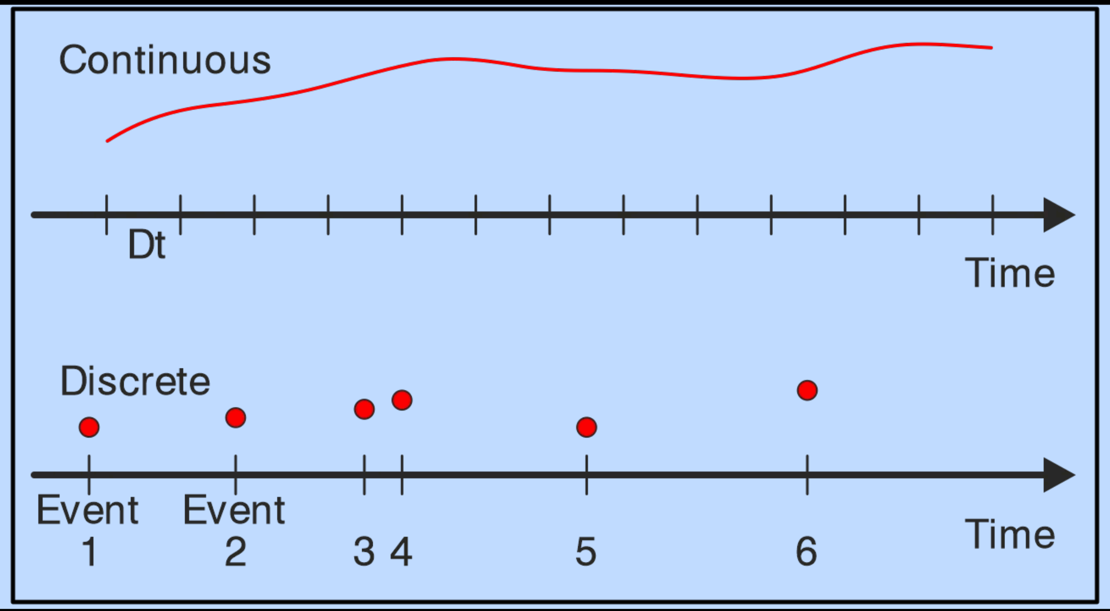
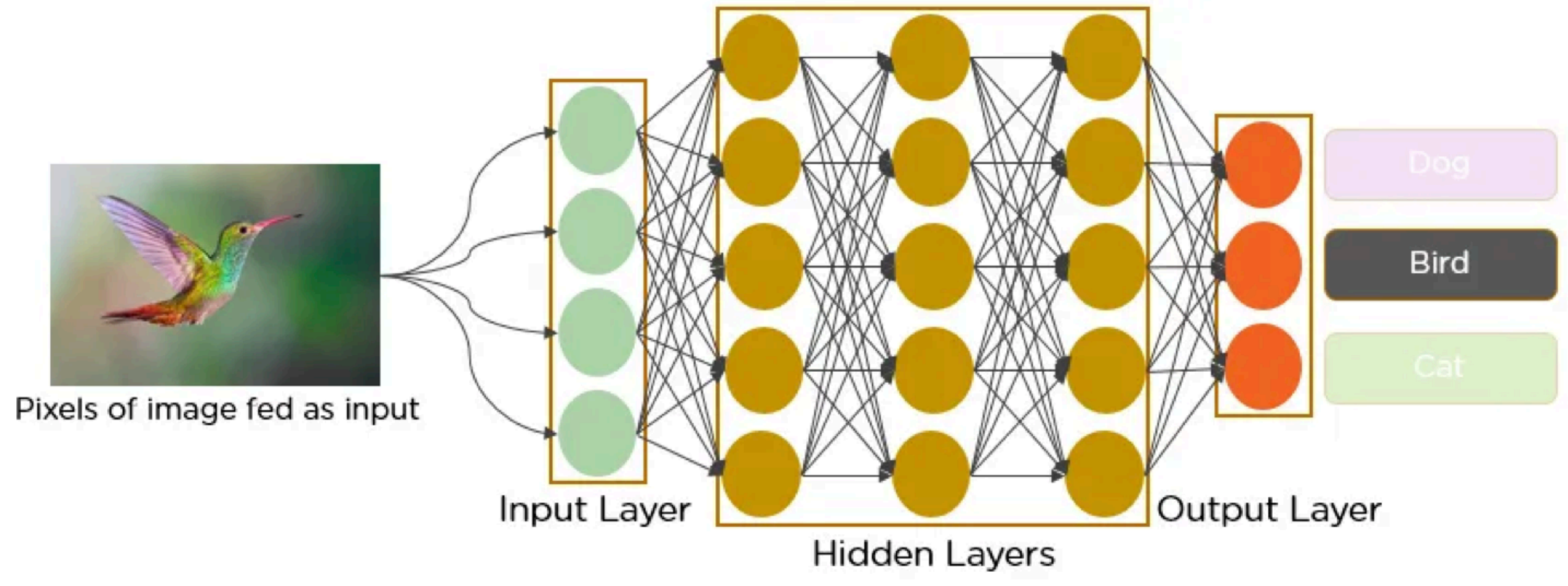
Paradigms for *Long* Time Series

Other Models

- Transformers, RNNs, CNNs etc. - specialized variants for capturing LRD, they still struggle to scale to very long sequences (>10000 steps)
  - Few hundred steps often already considered as long sequences
- ***Transformers***: self-attention!
  - Global context + **Positional encoding**
  - **Scalability and parallelization:** process the entire sequence simultaneously + handling of longer sequences
  - Quadratic self-attention complexity
- **CTMs, RNNs, CNNs:** all have their problems but also their strengths -> illustrated on the next slide

Traditional Models

& their Problems to Capture LRD

Continuous Time Models (CTMs)	Recurrent Neural Networks (RNNs)	Convolutional Neural Networks (CNNs)
	• Feed forward + backpropagation • Recurrence relation: one step to next	
• Model **underlying continuous process** of data • Capture inductive bias of data better (**irregular sampling, missing data**)	• Good for **stateful** settings like reinforcement learning/auto-aggressive tasks	• **Parallelizable** • Scale much better/ easier to train, • No vanishing gradient problem
• Complex (in handling irregular time intervals) • **Ineffcient/slow** • **Vanishing gradients** • **Memory** constraints	• **Vanishing gradient** (influence of earlier inputs diminishes exponentially) • Memory Capacity/**inefficient/slow** (not parallelizable)	• **Inefficient** inference • Slow • **Bounded context** (unable to address long dependencies) • Positional Bias

Picture CTM: https://media.softwaresim.com/Figure_1_-_Updated_State_over_simulated_time_in_continuous_and_discrete_simulation_wm2o9v.webp
Picture CNN: https://cdn.analyticsvidhya.com/wp-content/uploads/2024/08/416511-66c706889f0e2.webp

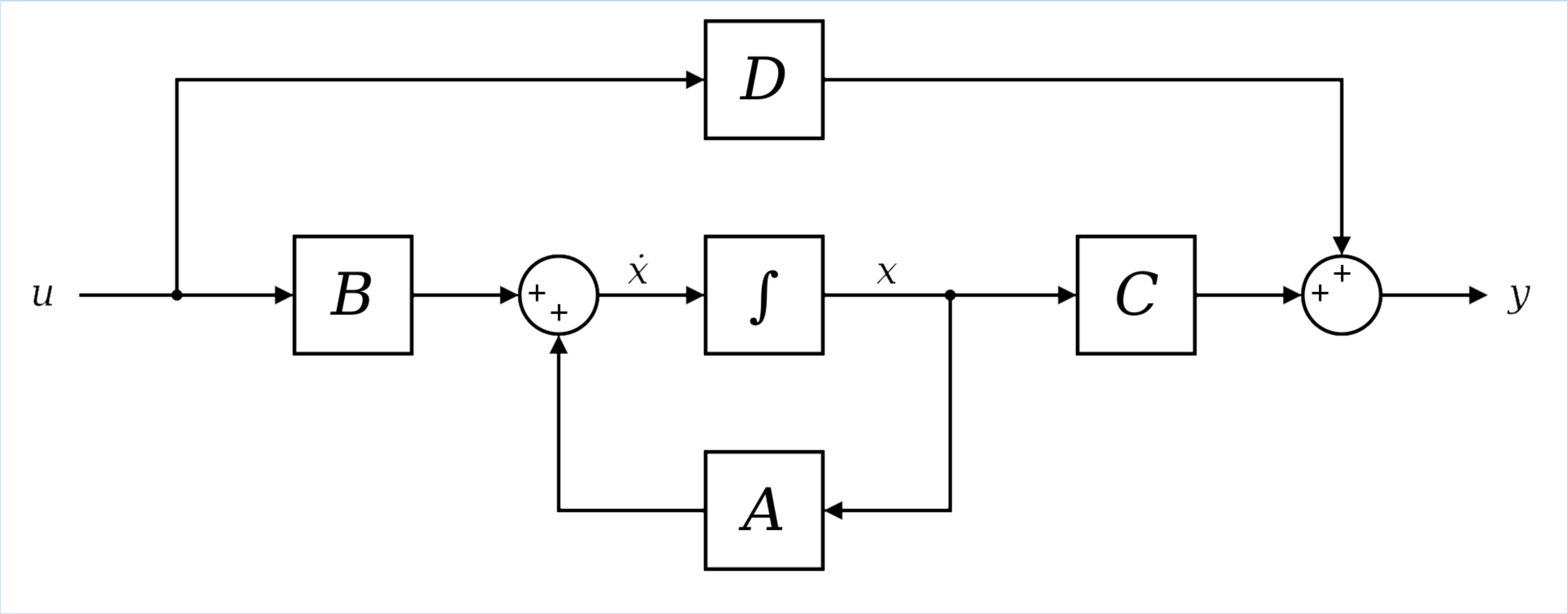
CTM, RNN, CNN

How do they Help us?

- All struggle with long sequences
- But combining their strength -> **State Space Model** -> **S4**
- Three different views/representations

Introduction SSM

What are State Space Models?

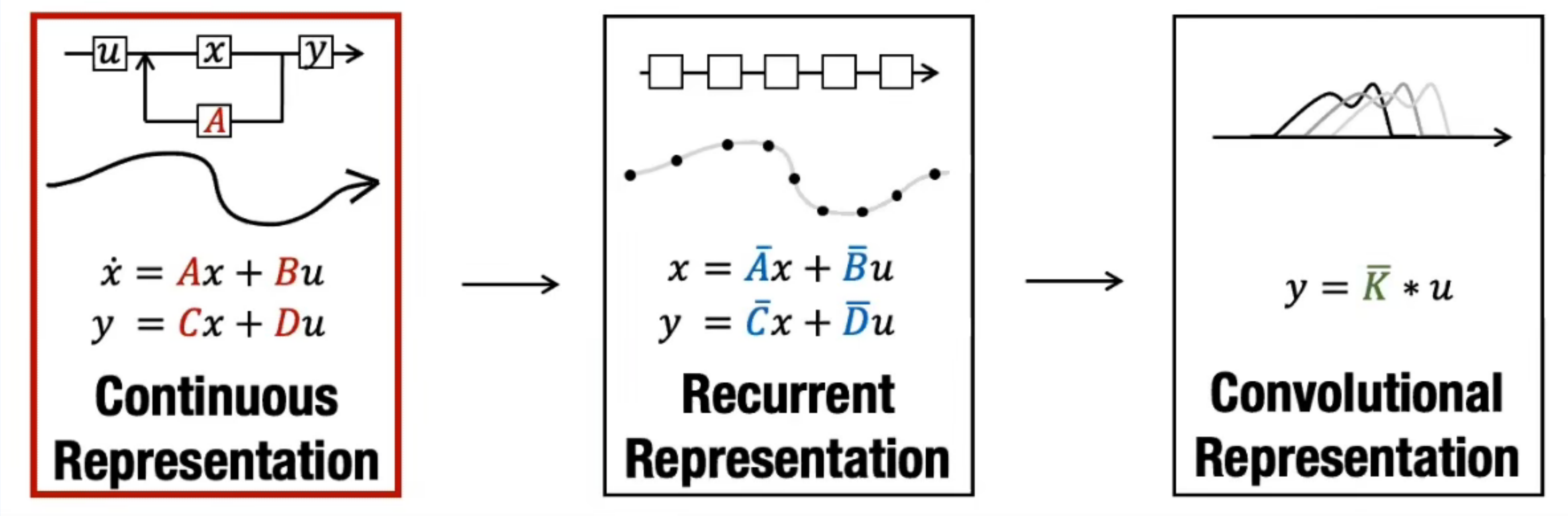


- You might know HMMs (Hidden Markov Models)?
- Continuous number of States
- Sequential Model (text sentences, time-series,…)
    - Data carry some dependency
- Sampled over continuous time (irregular sampling intervals)
- Used in fields such as control theory, computational neuroscience etc.
- Not been applicable to deep learning (theoretical reasons)

Picture: https://en.wikipedia.org/wiki/State-space_representation

State Space Model

All three Representations:



Picture: https://arxiv.org/pdf/2111.00396

SSM: Continuous Representation

Basis of the SSM

- Mostly theoretically
- Four learnable matrices: A, B, C, D (by gradient descent)
- Three variables that depend on time t: x, u, y
- Continuous: SSM maps function to function (u(t) -> y(t))
    - Benefits: functions more general than sequences -> always discretizable

how the state vector changes

state matrix

state vector/ latent state

input matrix

$$x'(t) = \boxed{A} \cdot x(t) + B \cdot u(t) \rightarrow \text{State equation}$$

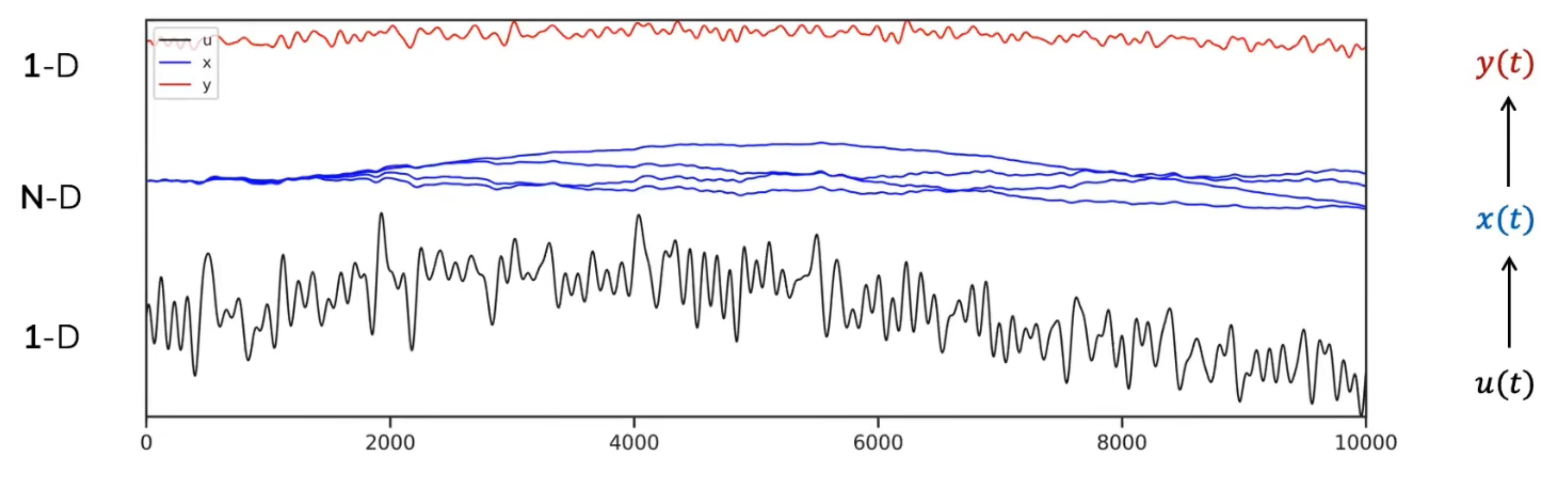
$$y(t) = C \cdot x(t) + D \cdot u(t) \rightarrow \text{Output equation}$$

output vector

output matrix

feed-forward matrix

input vector

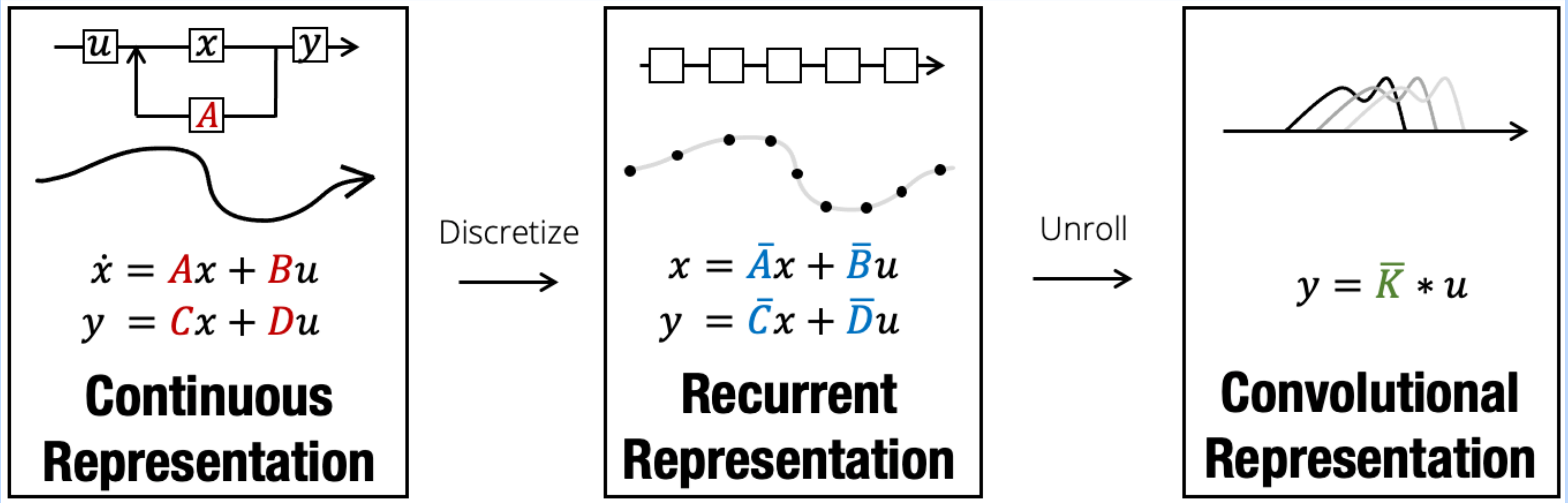


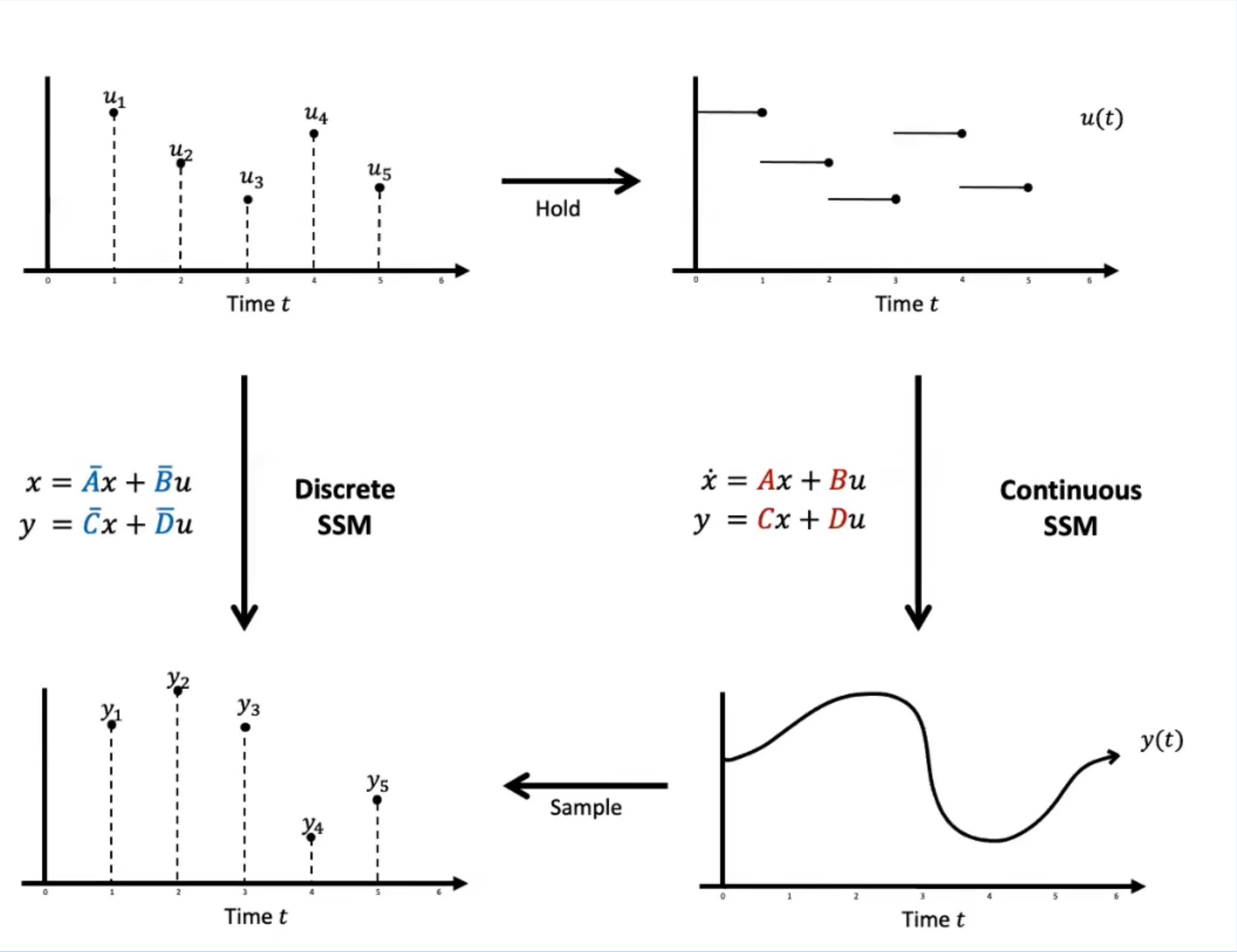
Picture: https://arxiv.org/pdf/2111.00396

Discretization!

One of the most Important Points in SSM

- Enables us to pass to the other two views
- Sequence to sequence map





Picture: from blog post « Structured State Spaces: Combining Continuous-Time, Recurrent, and Convolutional Models » by Albert GU et al. (2022)

SSM: Recurrent View

Real World Data comes in Form of Sequences

- Discretize:
  A, B, C, D -> $\bar{A}, \bar{B}, \bar{C}, \bar{D}$

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

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

- Δ: Step size (can be varying), resolution of the input
- Allowing the discrete SSM to be computed like an RNN
  - Autoregressive computation of state (recurrence)
- Not practical for training on modern hardware due to its sequentiality (GPUs/TPUs: need parallelization to be efficient)
- Solution = Convolutional View

Picture: https://arxiv.org/pdf/2111.00396

SSM: Convolutional View

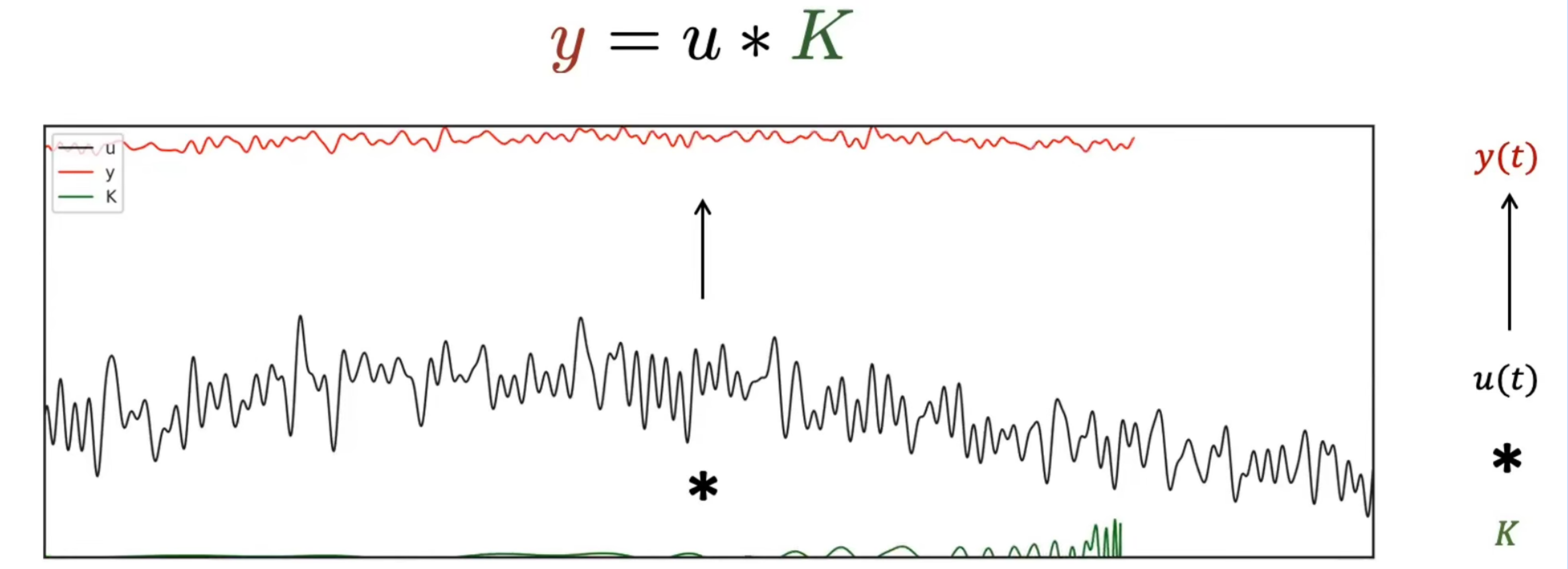
Unroll Linear Recurrence in Closed Form

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

$$\overline{\boldsymbol{K}} \in \mathbb{R}^L := (\overline{\boldsymbol{C}}\overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}\overline{\boldsymbol{B}}, \dots, \overline{\boldsymbol{C}}\overline{\boldsymbol{A}}^{L-1}\overline{\boldsymbol{B}})$$

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

- Most important representation
- Linear recurrences can be computed in parallel as a convolution
- First equation: can be computed very efficiently with FFTs, provided that $\bar{K}$ is known
- $\bar{K}$: SSM **convolution kernel** (same length as sequence), explicit formula parameterized in this special view using parameters A, B, C
    - Non-trivial
    - Focus of next part lies on the computation of $\bar{K}$

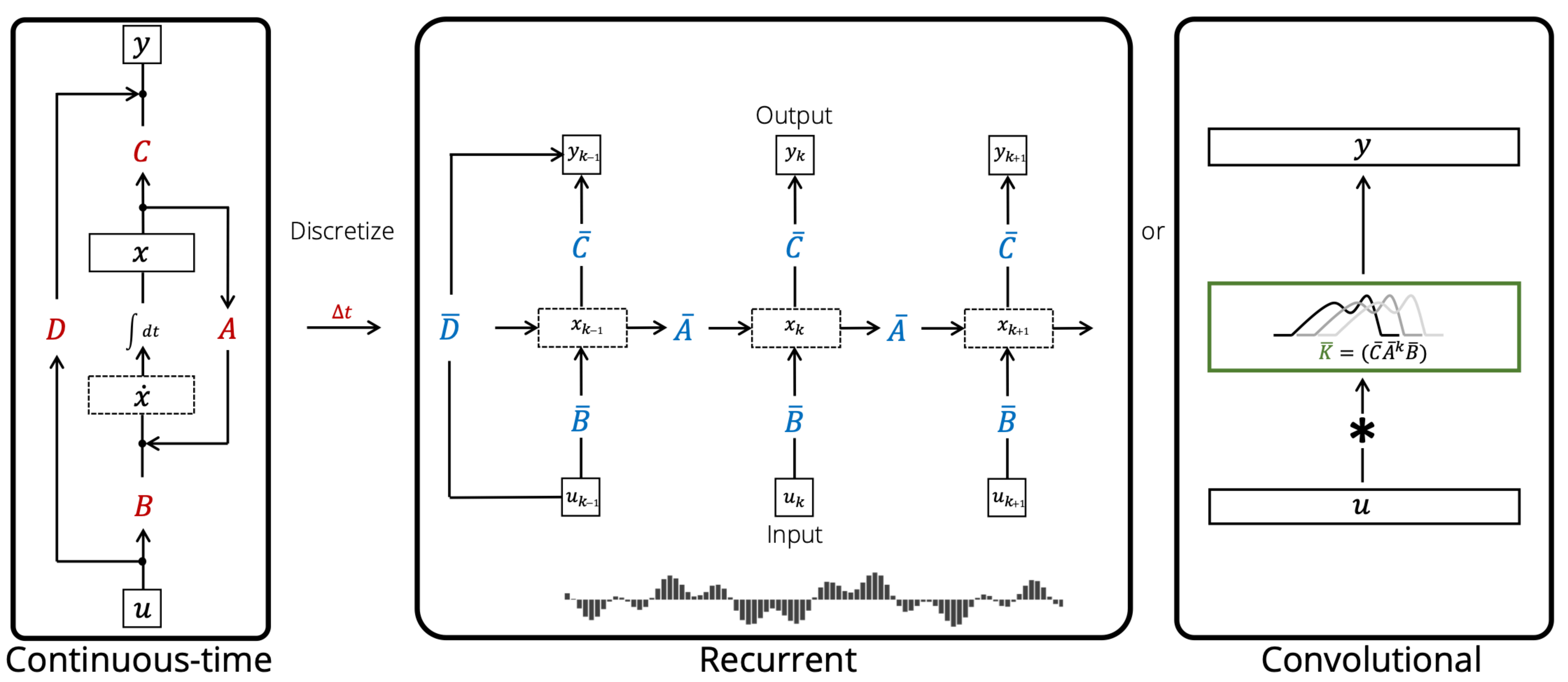


Structured State Spaces

Pros + Cons of Different Views

Continuous time	Recurrent	Convolutional
• Automatically handles **continuous** data • **Mathematical** feasible analysis (building HiPPO) • Irregular sampling	• **Unbounded** context • Efficient **inference** (constant-time state updates)	• Local, interpretable features • Efficient (**parallelizable**) training
• **Slow** training + inference	• **Slow** learning (lack of parallelism) • **Vanishing/exploding gradient**	• Slowness in online or **autoregressive** contexts (must recalculate entire input for each new data point) • **Fixed context size**

Structured State Spaces



Picture: https://www.youtube.com/watch?v=EvQ3ncuriCM

From the SSM to the S4

For long-term Dependencies

how the state vector changes

state matrix

state vector/ latent state

input matrix

$$x'(t) = \boxed{A} \cdot x(t) + B \cdot u(t) \rightarrow \text{State equation}$$

$$y(t) = C \cdot x(t) + D \cdot u(t) \rightarrow \text{Output equation}$$

output vector

output matrix

feed-forward matrix

input vector

- SSM: performs poorly in practice
  - Inherit properties of CTM, RNN, CNN … including problems with LRDs
  - For appropriate choices of the state matrix A, system could handle long-range dependencies mathematically and empirically
  - SSMs have nice properties *provided that* representations $\bar{A}$ and $\bar{K}$ are known
- SSM + **HiPPO** + **Convolutional Kernel** = **S4**
- Basically just SSM with special formulas for A and B
  - Reparameterization of the state matrix A using low-rank and normal terms

S4: The HiPPO Operator

What is it?

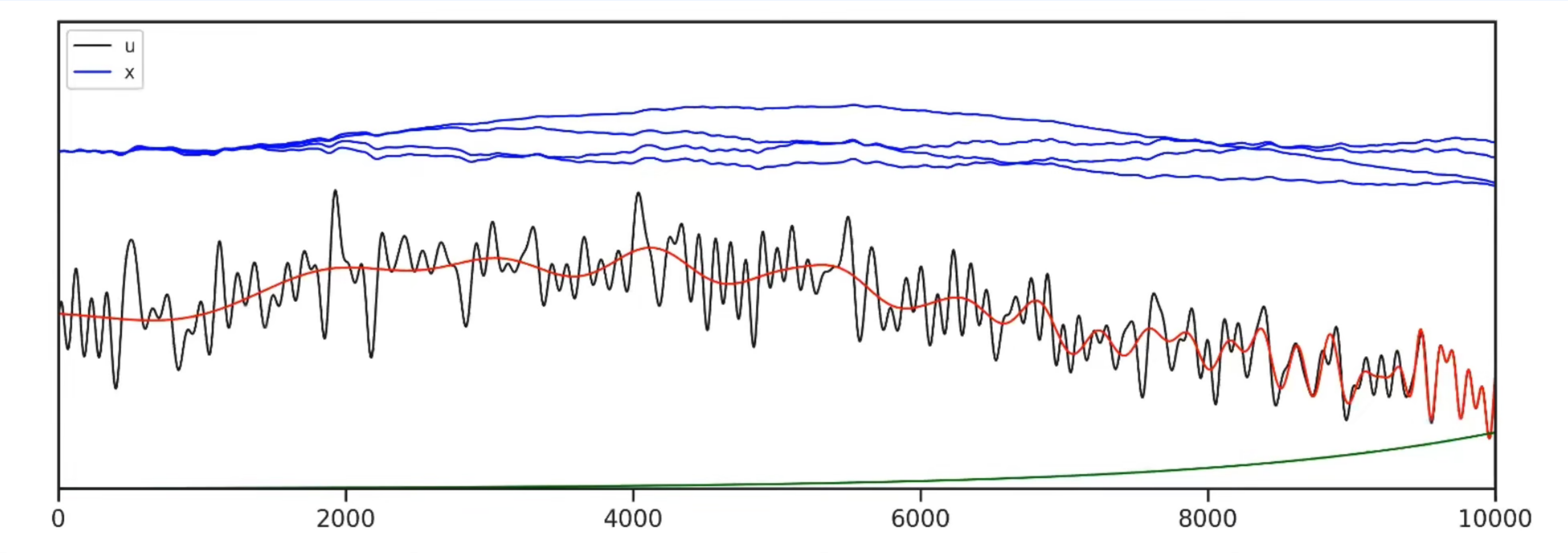
$$(\textbf{HiPPO Matrix}) \qquad A_{nk} = -\begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases}.$$

- **How can we remember context from millions of steps ago?**
  - Compress the past —> reconstruct the path
- **HiPPO** (= **Hi**gh-Order-**P**olynomial **P**rojection **O**perator)
- Continuous-time memorization: allows the state x(t) to memorize the history of the input u(t)
- Produces a hidden state that memorizes its history
- A is the more important matrix
  1. We only need to calculate it once
  2. It has a nice, simple structure

Picture: https://arxiv.org/pdf/2111.00396

S4: HiPPO

How does it look like?



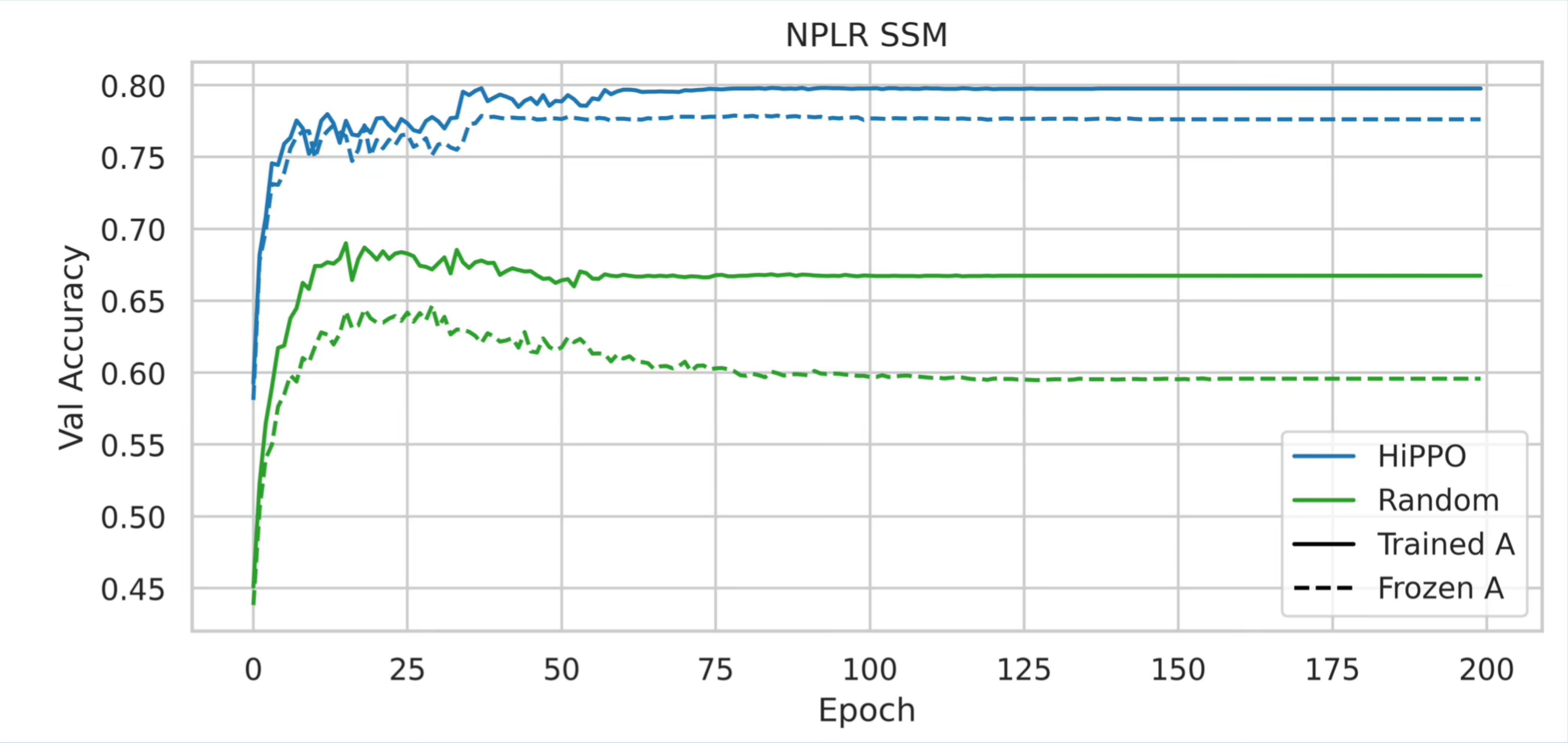
- Pass function u (black line) through HiPPO -> gives line x (blue line)
- Red line: approximation
  - Reconstruction of input (here: exponential approximation with 64 (coefficients) << 10000 hidden units)
  - Very accurate for recent past, decays over time
  - Always maintaining some information about past
- Green line: measures quality of approximation over time (exponentially)

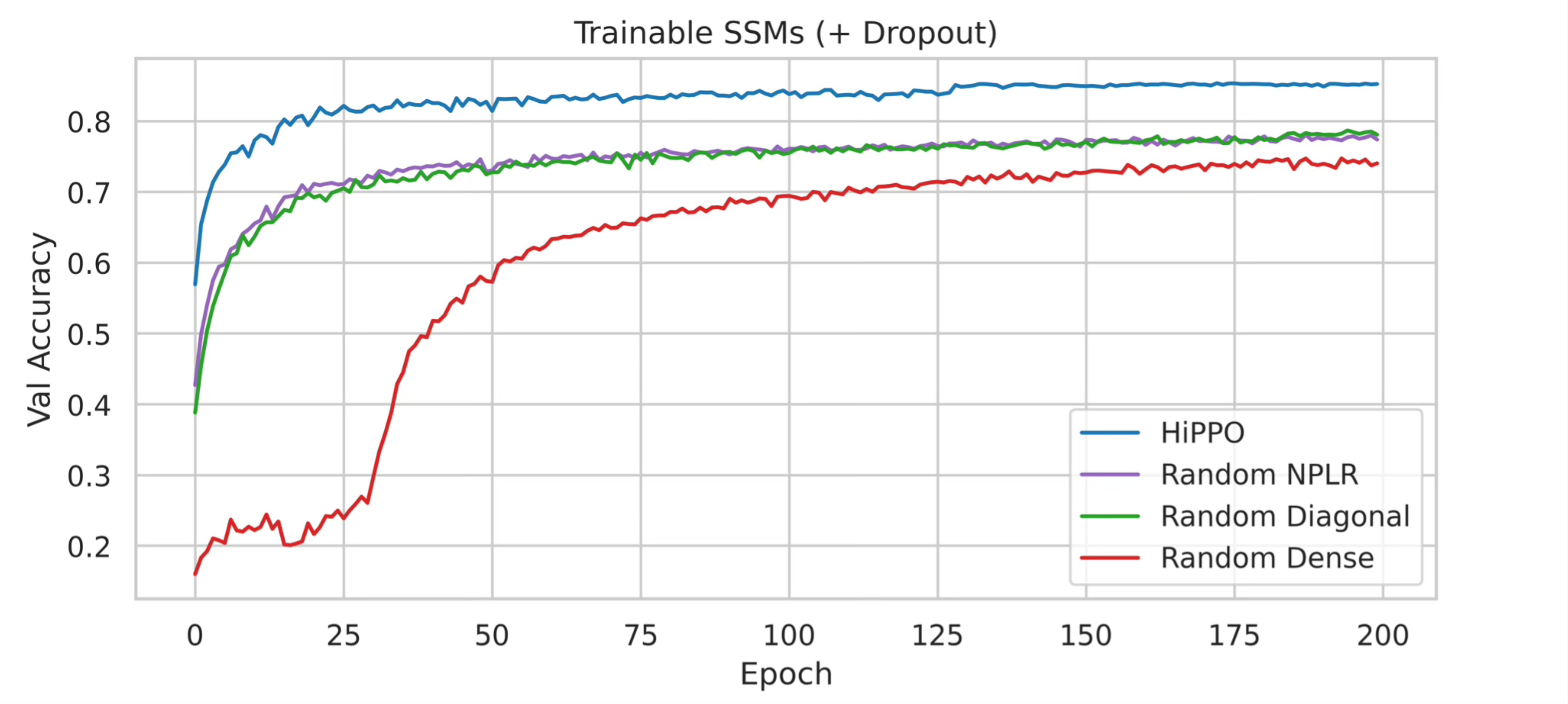
Picture: https://www.youtube.com/watch?v=luCBXCErkCs

S4: HiPPO

Crucial for Handling LRD

- Maintain a compressed summary of the entire history of a sequence
- **Memory Retention**: Prioritizes recent inputs to combat vanishing gradients
- **Orthogonality**: Ensures stable learning through orthogonal basis functions
- **Efficiency**: Low-rank structure





Pictures: https://arxiv.org/pdf/2111.00396

S4: Structured State Spaces

Issue

$$\overline{K} = (\overline{CB}, \overline{CAB}, \dots, \overline{CA}^{L-1}\overline{B})$$

- Computing convolution (fast) but convolution kernel (expensive)
- Convolutional kernel non-trivial
  - Powering up A -> $O(N^2L)$ operations and $O(NL)$ space
  - Too slow!
  - HiPPO: A -> $O(N + L)$ computation and memory usage
  - L: sequence length, N: number of states

Picture: https://arxiv.org/pdf/2111.00396

S4 Convolution Kernel

Solution!

- S4 kernel: helps to further reduce the runtime (three new techniques)
  - Instead of expanding the standard SSM in coefficient space -> compute its truncated generating function in frequency space
  - Huge improvement
- Computation very complicated

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 $\boxed{5}$)

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

Picture: https://arxiv.org/pdf/2111.00396

Structured State Spaces (S4)

Wrap up: What is S4? Key Innovation?

- New parameterization + computation using a Cauchy kernel
- Enhances S4's ability to handle sequences with thousands of time steps without significant computational overhead
- Constructed to not forget things

Experiments - S4

Large-scale Generative Modeling + Fast Autoregressive Generation

- **CIFAR-10:** autoregressive models
- No 2D inductive bias
- Competitive with the best models designed for this task

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×)

- **WikiText-103**: language modeling
- Approaches performance of transformers with much faster generation

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×)**

Pictures: https://arxiv.org/pdf/2111.00396

Experiments - S4

Speech Classification 1.1

- Irregular continuous data
- Missing values
- Can adapt to any sampling rate (different frequencies) at test time by simply changing its step size
- WaveGAN-D: CNN, second best on Raw, but cannot deal with different sampling rate

		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**

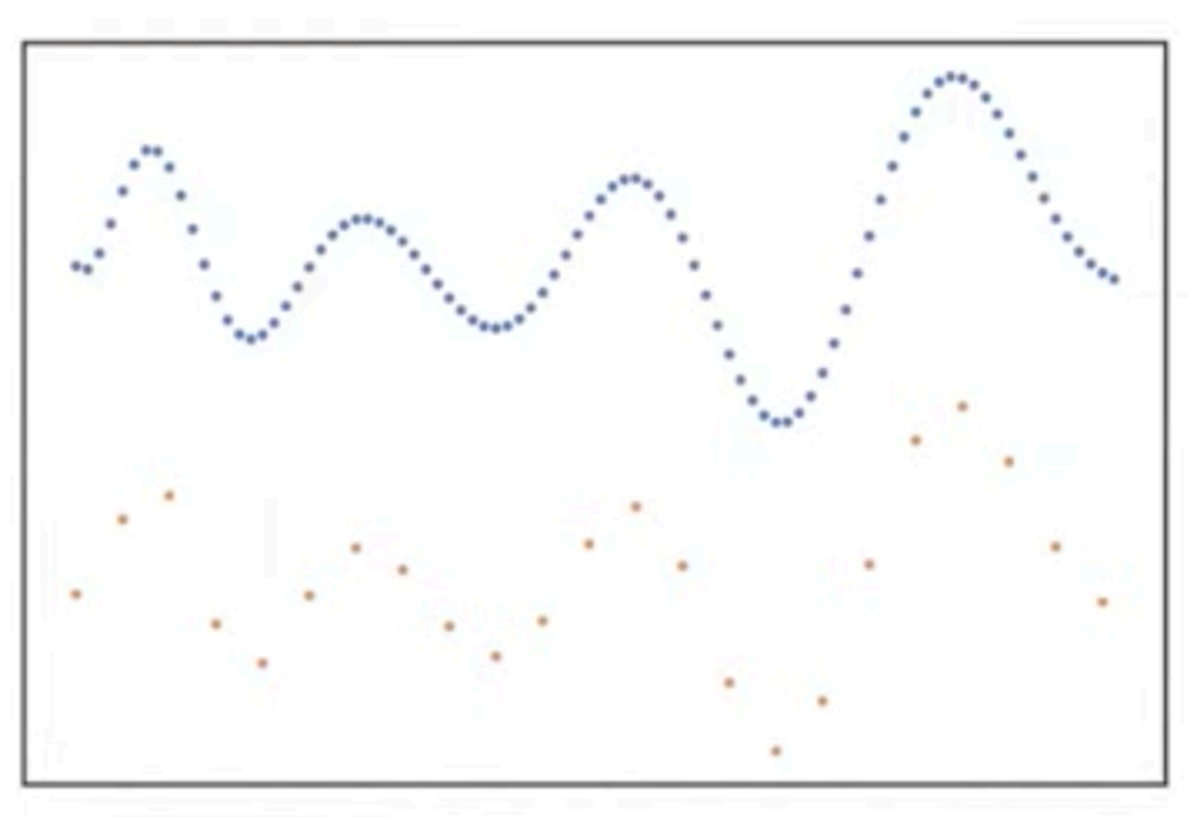


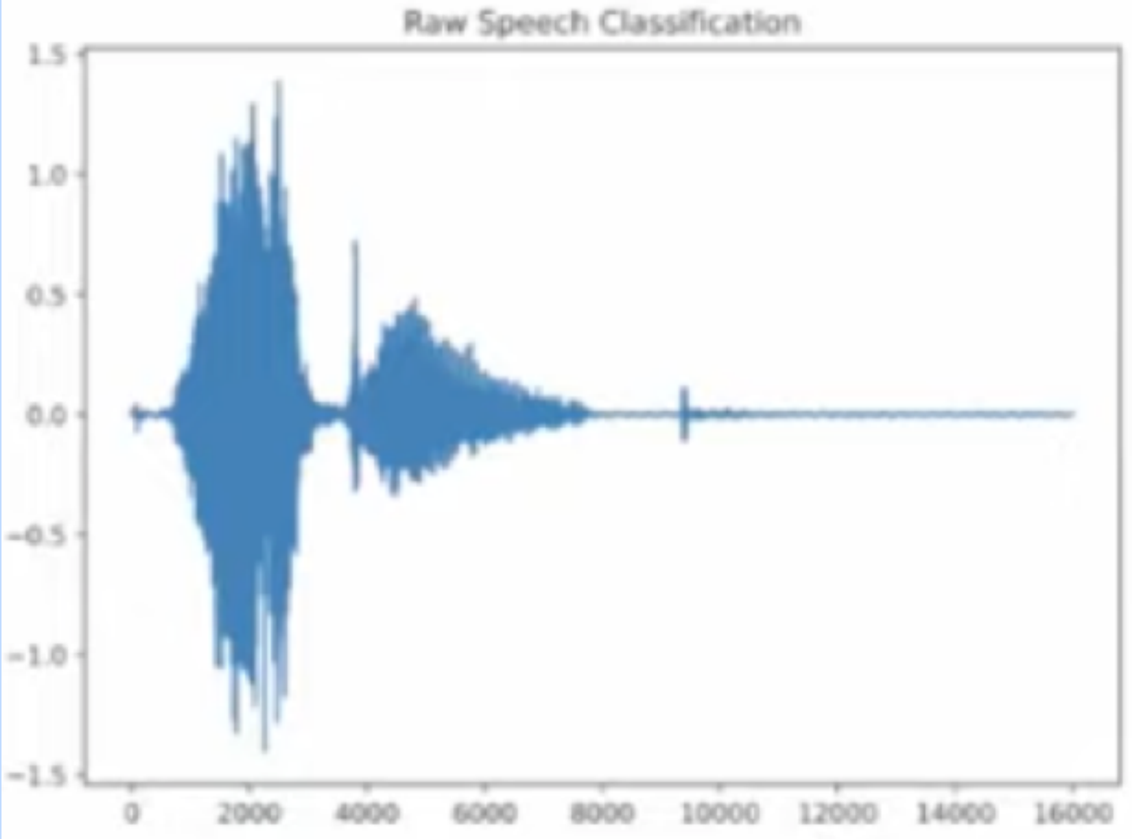
Table: https://www.youtube.com/watch?v=luCBXCErkCs

Experiments - S4

Speech Classification 1.2

- 1.7% error on length-16000 sequences

Raw data requires specialized CNNs



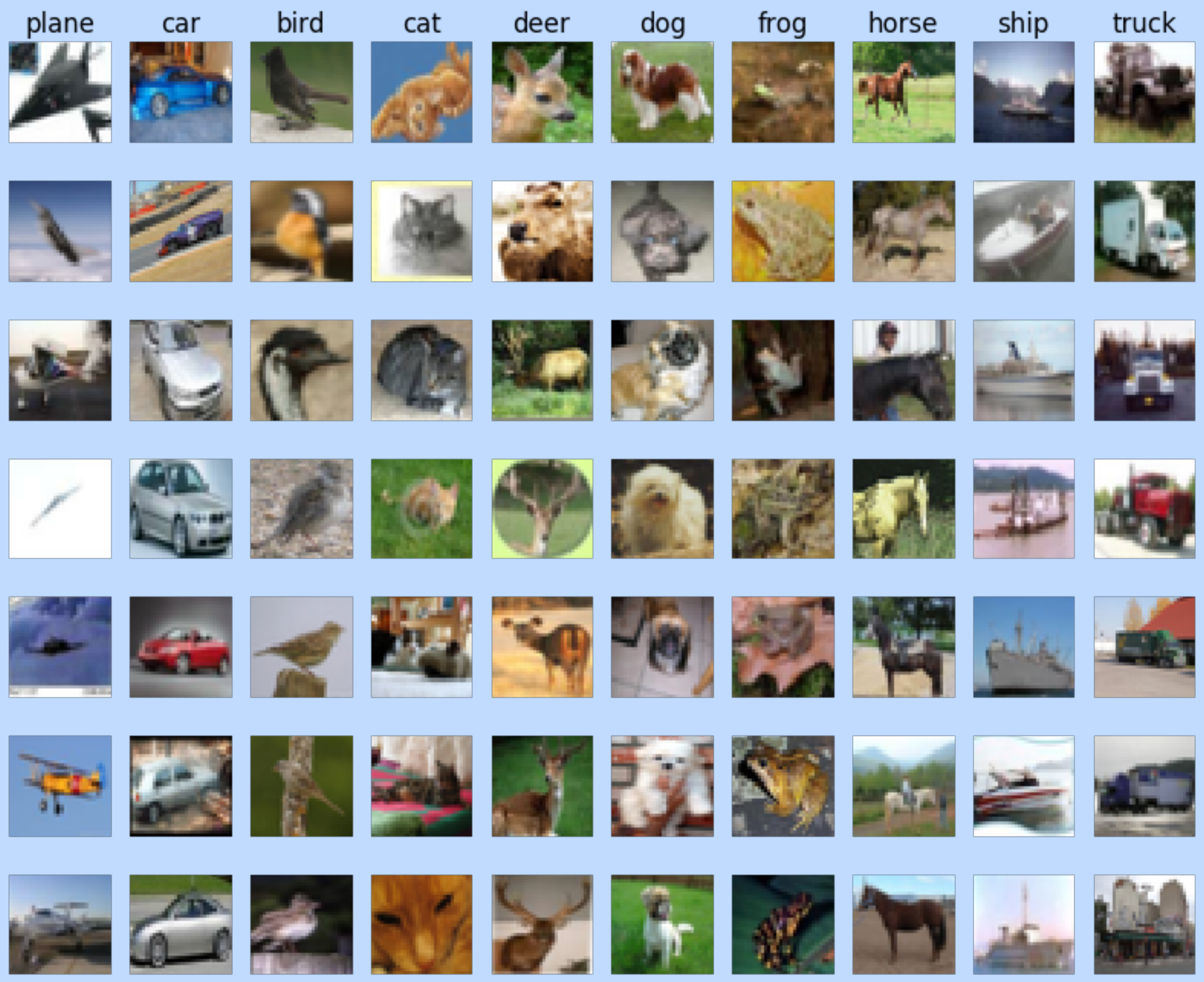
	L=160	L=16000	
	MFCC	RAW	
Transformer	90.75	✗	
Performer	80.85	30.77	
ODE-RNN	65.9	✗	
NRDE	89.8	16.49	
ExpRNN	82.13	11.6	
LipschitzRNN	88.38	✗	
CKConv	**95.3**	71.66	
WaveGAN-D	✗	96.25	← 88x larger than S4
LSSL	93.58	✗	
S4	93.96	**98.32**	

Table: https://www.youtube.com/watch?v=luCBXCErkCs

Experiments - S4

Sequential Image Classification

	sMNIST	pMNIST	sCIFAR	
Transformer	98.9	97.9	62.2	Transformers
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	RNNs
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	CNNs
CKConv	99.32	98.54	63.74	
LSSL	99.53	**98.76**	84.65	SSMs
S4	**99.63**	98.70	**91.13**	



- Perform 15-30% better than all previously evaluated sequence models

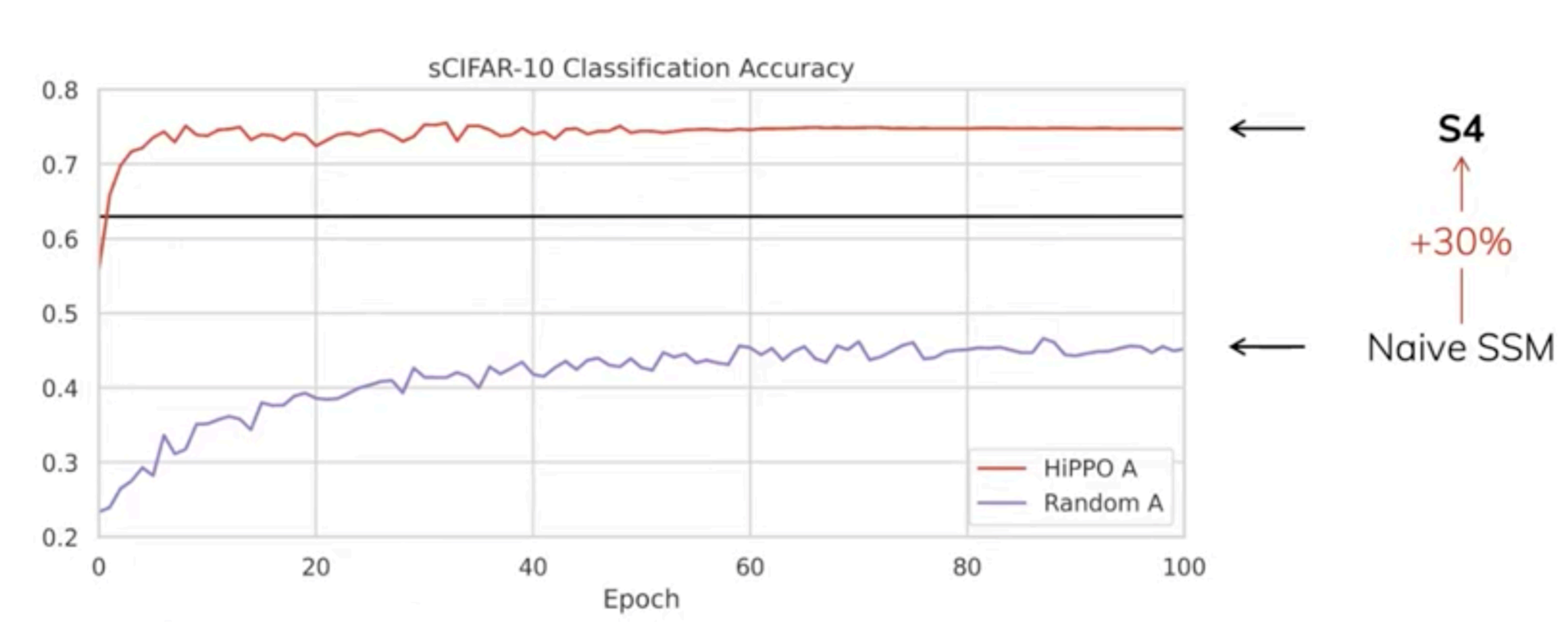
- 91% accuracy on sequential CIFAR-10

Table: https://arxiv.org/pdf/2111.00396
Picture: https://www.mabl.com/hubfs/Screen Shot 2018-01-25 at 5.00.49 PM.png

Experiments - S4

The Importance of HiPPO

- Black line: Transformer
- Typically in deep learning: randomly initializing all the parameters -> it does terribly
- Plug in formula for matrix: from much below the baseline to substantially above!!



Picture: https://www.youtube.com/watch?v=luCBXCErkCs

Experiments - S4

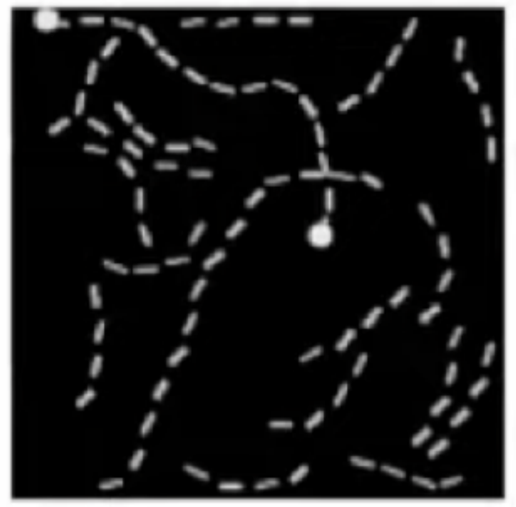
Long Rage Arena Benchmark

- Six tasks, different types of data from 1000-16000 length
- 88% accuracy on Path-X (first model!)

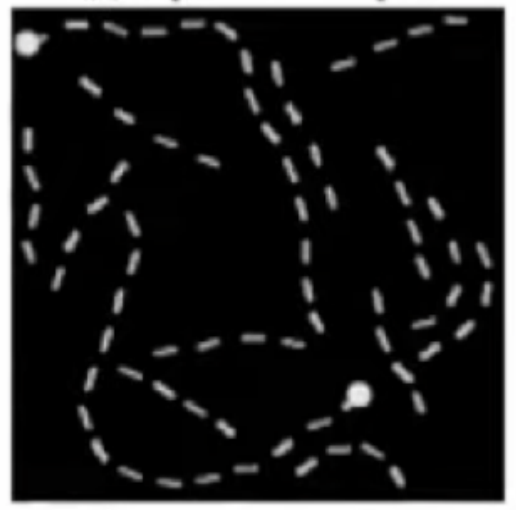
Benchmark spanning text, images, symbolic reasoning (length 1K-16K)

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.

Table: https://arxiv.org/pdf/2111.00396

Experiments - S4

Take-Away: Towards a General-purpose Sequence Model

- **Large-scale generative modeling** (competitive with the best autoregressive models)
- **Fast autoregressive generation** (perform 60× faster pixel/token generation)
- **Sampling resolution change** (adapt to changes)
- **Learning with weaker inductive biases** (surpasses Speech CNNs on speech classification, matches a 2-D ResNet on sequential CIFAR with over 90% accuracy)

.

Further Application for S4 Model

Some Examples

- Audio generation
- Large scale audio pre-training
- S4 extensions/variations
- S4 + Transformers for language models
- 2D + 3D versions of S4 (images, video)
- …

Their Conclusion

Outlook

- S4 **highly versatile**, since it can be applied to text, vision, audio and time-series tasks (or even graphs)
    - Transformers are still dominating language modelling (S4 is more for continuous data)
- S4 **combined** with other sequence models to complement their strengths
- Challenge: to know when to favor one view over another, depending on stage of process (training or inference) + the type of data
- Ability to handle **very long sequences**, generally with a **lower number of parameters** than other models (ConvNet or transformers), while still being very fast
- Much **potential**, promising ideas

THANK YOU VERY MUCH! :)

Do you have any questions?