

MODELING SEQUENCES WITH STRUCTURED STATE SPACES

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER
SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Albert Gu
June 2023

© 2023 by Albert Gu. All Rights Reserved.
Re-distributed by Stanford University under license with the author.



This work is licensed under a Creative Commons Attribution-
Noncommercial 3.0 United States License.
<http://creativecommons.org/licenses/by-nc/3.0/us/>

This dissertation is online at: <https://purl.stanford.edu/mb976vf9362>

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Chris Re, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Percy Liang

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Scott Linderman

Approved for the Stanford University Committee on Graduate Studies.

Stacey F. Bent, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format.

Abstract

Substantial recent progress in machine learning has been driven by advances in sequence models, which form the backbone of deep learning models that have achieved widespread success across scientific applications. However, existing methods require extensive specialization to different tasks, modalities, and capabilities; suffer from computational efficiency bottlenecks; and have difficulty modeling more complex sequential data, such as when long dependencies are involved. As such, it remains of fundamental importance to continue to develop principled and practical methods for modeling general sequences.

This thesis develops a new approach to deep sequence modeling using state space models, a flexible method that is theoretically grounded, computationally efficient, and achieves strong results across a wide variety of data modalities and applications. First, we introduce a class of models with numerous representations and properties that generalize the strengths of standard deep sequence models such as recurrent neural networks and convolutional neural networks. However, we show that computing these models can be challenging, and develop new classes of *structured state spaces* that are very fast on modern hardware, both when scaling to long sequences and in other settings such as autoregressive inference. Finally, we present a novel mathematical framework for incrementally modeling continuous signals, which can be combined with state space models to endow them with principled state representations and improve their ability to model long-range dependencies. Together, this new class of methods provides effective and versatile building blocks for machine learning models, especially towards addressing general sequential data at scale.

Acknowledgements

First of all, I am grateful to my advisor Christopher Ré, whose support and guidance throughout my PhD has helped me grow immensely. Beyond fundamentals of research and technical story-telling which he is particularly good at, perhaps the most important value I acquired from Chris is a deep sense of intellectual curiosity and drive to understand and explore, which transformed me tremendously between the beginning and end of my PhD. I would also like to thank the rest of my reading and orals committee—Percy Liang, Scott Linderman, Tatsu Hashimoto, and Andrea Montanari—who have been very accommodating through many last minute appointments and always helpful in general.

Next, I want to thank my lab and all my mentors and collaborators throughout the years. I've benefitted greatly from the wisdom and experience of senior students and postdocs including Chris De Sa, Virginia Smith, Ioannis Mitliagkas, Stephen Bach, Sharon Li, Fred Sala, and Alex Ratner. I've been fortunate to collaborate with incredibly talented students and researchers; I should give special thanks to those that have worked closest with me and had to put up with my manic work style, including Fred Sala, Ines Chami, Khaled Saab, Nimit Sohoni, Megan Leszczynski, Anna Thomas, Beliz Gunel, Jared Dunnmon, Vaggos Chatziafratis, Jared Davis, and Dat Nguyen.

Among collaborators, there are a few that I want to single out. First, Atri Rudra, along with his students Isys Johnson, Aman Timalsina, Matthew Eichhorn, and Amit Blonder, has been a constant from the beginning of my PhD to the end, and has played an important role in many of the results on the line of work in this thesis. I deeply admire his infinite patience and good nature through my flaky scheduling and intense deadlines, as well as his willingness to work on any half-baked theory problem I throw at him. Tri Dao has been one of my closest collaborators and best friends throughout the years. Looking back, I'm immensely proud of how we've struggled and grown together the whole time, into independent researchers who have made accomplishments in different directions that we can

be very proud of. I still fondly remember the early years when we barely knew anything and tried learning together through classes and two-person reading groups... but mostly just worked out and configured Vim/Emacs. Karan Goel was instrumental in the development of research toward the later part of my PhD, including many of the results in this thesis. He has been endlessly willingly to entertain my crazy ideas, usually developed two weeks before deadlines, and join me on sleepless sprints pushing for the next “drive-by SOTA”. And of course he’s also been one of my best friends – here’s to endless Robots and Hunan.

Beyond collaborations, the rest of the lab have always been great friends to me, including Avner May, Fred Sala, Nimit Sohoni, Beidi Chen, Dan Fu, Ines Chami, Paroma Varma, Mayee Chen, Simran Arora, Gordon Downs, Michael Zhang, Arjun Desai, Sabri Eyuboglu, Neel Guha, Gautam Machiraju, Avanika Narayan, Eric Nguyen, Vishnu Sarukkai, Brandon Yang, Laurel Orr, Megan Leszczynski, Sarah Hooper, Jian Zhang, Jared Dunnmon, Alex Ratner, Braden Hancock, Stephen Bach, Peng Xu, and Chris De Sa. (That should be mostly everyone, although Chris has had so many students and I’ve been here so long that it’s hard to keep track...)

Outside of the lab, I’m tremendously lucky to have had a strong support network of friends and family. Within Stanford, I’ve made many close friends and acquaintances throughout the years. Special thanks to Ananya Kumar, Michael Xie, Shiori Sagawa, Minae Kwon, Karan Goel, Shreya Rajpal, Kristy Choi, Gary Cheng, Daniel Levy, Chris Cundy, and more, for being my “second cohort” when I started feeling more comfortable and tried to integrate into the Stanford community. Luckily I’ve delayed long enough that you’re also graduating soon, and I’m looking forward to our next adventures. My friends from college and high school have always provided a source of comfort away from the pace of Stanford: thanks to Bo Li, Melissa Huang, Ivan Wang, Joy Lin, Andrew Xiao, Weikun Liang, Hank Hwang, Annie Chen, Jo Chen, Amol Aggarwal, David Zeng, and many more, for all the fun hangouts, food, and games. I am grateful to Winnie Yeung for your patience and support, including helping with my work-life balance by planning fun trips and vacations, our shared fondness of sports, and encouraging my In-n-Out and Popeye’s habit. I must also thank Anny Ni for being my rock through the difficult early years of my PhD; I would not have made it without you.

Finally, I’m most grateful to my parents, Jitang Gu and Angela Liang, for their unconditional love, support, and understanding.

Contents

Abstract	iv
Acknowledgements	v
1 Introduction	1
1.1 Deep Sequence Models	1
1.2 State Space Sequence Models	4
1.2.1 A General-Purpose Sequence Model	5
1.2.2 Efficient Computation with Structured SSMs (S4)	6
1.2.3 Addressing Long-Range Dependencies with HIPPO	7
1.2.4 Applications, Ablations, and Extensions	8
1.3 Thesis Notes	9
1.3.1 Bibliographic Remarks	10
1.3.2 Notation and Conventions	11
I Structured State Spaces	13
2 Sequence Modeling with State Space Models	14
2.1 Background: The Sequence Model Framework	14
2.1.1 Learning with Deep Sequence Models	15
2.1.2 Example: Regression and Classification	16
2.1.3 Example: Autoregressive Generative Modeling	17
2.2 Background: State Space Models	18
2.2.1 Linear Time Invariant (LTI) SSMs	19
2.3 State Space Sequence Models	20
2.3.1 The Continuous Representation (Discretization)	20

2.3.2	The Recurrent Representation (Efficient Inference)	22
2.3.3	The Convolutional Representation (Efficient Training)	23
2.3.4	Summary of SSM Representations	25
2.3.5	A Note on SSM Dimensions	25
2.4	Interpreting SSM Representations	27
2.4.1	SSMs as Continuous Models	27
2.4.2	SSMs as Recurrences	29
2.4.3	SSMs as Convolutions	31
2.5	Discussion: Naming Conventions for SSMs	33
2.5.1	Classical State Space Models	33
2.5.2	State Space Sequence Models	33
2.5.3	Deep State Space Models	34
3	Computing Structured SSMs	36
3.1	Motivation: Computational Difficulty of SSMs	37
3.1.1	Discussion: General Recurrent Computation	38
3.1.2	Discussion: General Convolutional Computation	38
3.1.3	Structured SSMs	39
3.2	Diagonally Structured State Space Models	40
3.2.1	S4D: Diagonal SSM Algorithms	41
3.2.2	Complete Implementation Example	42
3.3	The Diagonal Plus Low-Rank (DPLR) Parameterization	43
3.3.1	Overview of the DPLR State Space Kernel Algorithm	44
3.3.2	S4-DPLR Algorithms and Computational Complexity	45
3.3.3	Hurwitz (Stable) DPLR Form	45
3.4	Additional Parameterization Details	47
3.4.1	Discretization	47
3.4.2	Parameterization of \mathbf{A}	48
3.4.3	Parameterization of \mathbf{B} and \mathbf{C}	48
3.4.4	Complex Numbers	48
3.4.5	Conjugate Symmetry	48
3.5	Comparing S4-Diag and S4-DPLR	49
3.5.1	Expressivity	49
3.5.2	Initialization of Parameters	50
3.5.3	Computational Complexities	50
3.5.4	Overall Parameterization	50

3.6	Discussion	51
3.6.1	Comparison between S4 and Other Sequence Models	51
3.6.2	Other Structured SSMs	52
II	Orthogonal State Space Models for Online Memorization	54
4	HIPPO: Continuous Memory with Optimal Polynomial Projections	55
4.1	Introduction	56
4.2	High-order Polynomial Projection Operators	57
4.2.1	HIPPO Problem Setup	58
4.2.2	General HIPPO framework	59
4.2.3	High Order Projection: Measure Families and HIPPO ODEs	61
4.2.4	HIPPO Recurrences: from Continuous- to Discrete- Time	61
4.2.5	Low Order Projection: Memory Mechanisms of Gated RNNs	62
4.3	LegS: Scaled Measures for Timescale Robustness	62
4.3.1	Timescale Robustness	63
4.3.2	Computational Efficiency	64
4.3.3	Gradient Flow	64
4.3.4	Approximation Error Bounds	64
4.4	Empirical Validation	65
4.4.1	Online Function Approximation and Speed Benchmark	65
4.4.2	Additional Analysis and Ablations of HIPPO Methods	66
4.4.3	Timescale Robustness of HIPPO-LegS	68
4.5	Conclusion	70
5	HIPPO as Orthogonal SSMs	71
5.1	Framework: Revisiting HIPPO as Orthogonal SSMs	72
5.1.1	Summary of HIPPO Matrices	72
5.1.2	Orthogonal State Space Models	73
5.1.3	Previous HIPPO Methods	75
5.1.4	The General Result	77
5.2	Generalizations of LegS	77
5.3	Finite Window TO-SSMs	78
5.3.1	HIPPO-FouT	78
5.3.2	Approximating Delay Networks	79
5.4	Properties of TO-SSMs: Timescales and Normalization	79

5.5	Summary and Discussion	82
5.5.1	New Methods and Visualizations	82
5.5.2	General HIPPO Operators	83
5.5.3	Interpretation and Initialization of TO-SSMs as Sequence Models	83
6	Combining Orthogonal and Structured State Space Models	85
6.1	Overview: Motivation and Partial Progress	86
6.1.1	A Resolution to Problem 1	86
6.1.2	An Attempt for Problem 2	87
6.2	DPLR Structure of HIPPO Matrices	88
6.2.1	HIPPO Cannot be Diagonalized	88
6.2.2	Normal Plus Low-Rank (NPLR) Forms of HIPPO	89
6.2.3	DPLR Form	89
6.2.4	Hurwitz DPLR Form	90
6.3	Diagonal Approximations of HIPPO	90
6.3.1	Forms of HIPPO-LegS	91
6.3.2	S4D Instantiations	92
6.4	Discussion: HIPPO vs. S4	94
6.4.1	Instantiations of HIPPO	95
6.4.2	Instantiations of S4	96
6.4.3	Combining S4 and HIPPO	96
6.4.4	Summary of Named S4 Variants	97
III	Applications and Extensions	98
7	Empirical Evaluation: S4 as a General Sequence Model	99
7.1	Summary of S4 Details	100
7.2	A Simple SSNN Architecture	101
7.2.1	Multiple Channels	101
7.2.2	Parameter Tying	101
7.2.3	Linear Mixing	102
7.2.4	Nonlinear Activations	102
7.2.5	Residuals and Normalization	102
7.2.6	Bidirectional Modeling	102
7.3	Empirical Results	103
7.3.1	Sequential Image Classification	104

7.3.2	Time Series Regression	104
7.3.3	Speech Classification	105
7.3.4	Long Range Arena (LRA)	108
7.3.5	Time Series Forecasting	109
7.3.6	Large-scale Autoregressive Generative Modeling	110
7.4	Additional Capabilities of S4	111
7.4.1	Continuous-time: Sampling Rate Adaptation	112
7.4.2	Recurrent: Efficient Inference	113
7.4.3	Convolutional: Efficient Training	114
7.4.4	Long-Range Dependencies	114
7.5	Ablations of the S4 and HIPPO Theory	115
7.5.1	Ablations Protocol	116
7.5.2	O-SSM Normalization Theory: Initialization of Δ and C	117
7.5.3	SSM Parameterization	118
7.5.4	S4D Initialization Ablations	118
7.5.5	Trainability of (A, B) Matrices	121
7.5.6	S4D vs. S4	122
7.5.7	DPLR Ablations: The Effect of HIPPO	122
7.5.8	Comparisons Between HIPPO Methods	124
8	SaShiMi: S4 for Audio Waveform Generation	128
8.1	Introduction	128
8.2	Related Work	131
8.3	The SaShiMi Architecture	132
8.4	Experiments	134
8.4.1	Unbounded Music Generation	135
8.4.2	Model ablations: Slicing the SaShiMi	135
8.4.3	Unconditional Speech Generation	137
8.5	Discussion	140
9	S4ND: Extending S4 to Multi-dimensional Signals	141
9.1	Introduction	141
9.2	Related Work	144
9.3	Method	145
9.3.1	S4ND	146
9.3.2	Resolution Change and Bandlimiting	147

9.4	Experiments	148
9.4.1	S4ND in 1D & 2D: Large-scale Image Classification	148
9.4.2	S4ND in 3D: Video Classification	149
9.4.3	Continuous-signal Capabilities for Images	152
10	Conclusion	155
10.1	Core Model Understanding and Improvements	156
10.2	Application Areas	158
10.3	Hardware Efficiency	159
IV	Appendices	161
A	Technical Preliminaries	162
A.1	Discretization	162
A.1.1	Overview	162
A.1.2	Picard Iteration	163
A.1.3	Numerical Integration Methods	164
A.1.4	Discretization of LTI Systems (T-SSMs)	164
A.1.5	Discretization of Linear Scale Invariant (LSI) Systems	166
A.2	Orthogonal Polynomials	167
A.2.1	Properties of Legendre Polynomials	168
A.3	Other Mathematical Identities	169
A.3.1	Leibniz Integral Rule	169
A.3.2	Woodbury Matrix Identity	170
B	Appendix for Chapter 2	171
B.1	Proofs: RNNs are SSMs	171
B.1.1	Intuition and Proof Sketches	172
B.1.2	Capturing Gates through Discretization	173
B.1.3	Capturing Nonlinear Dynamics through Picard Iteration	174
B.1.4	Capturing Deep, Linear, Gated RNNs	176
C	Appendix for Chapter 3	178
C.1	S4-DPLR Algorithm Details	178
C.1.1	Computing the S4 Recurrent View	178
C.1.2	Computing the Convolutional View	180

C.2 Computation Complexity Comparison: SSM, S4, and Other Sequence Models	185
D Appendix for Chapter 4	188
D.1 General HIPPO Framework	188
D.1.1 Measure and Basis	189
D.1.2 The Projection and Coefficients	191
D.1.3 Coefficient Dynamics: the HIPPO Operator	192
D.1.4 Discretization	193
D.2 Derivations of HIPPO Projection Operators	194
D.2.1 Derivation for Translated Legendre (HIPPO-LegT)	194
D.2.2 Derivation for Scaled Legendre (HIPPO-LegS)	197
D.3 HIPPO-LegS Theoretical Properties	200
D.3.1 Timescale equivariance	200
D.3.2 Speed	200
D.3.3 Gradient Norms	201
D.3.4 Function Approximation Error	203
D.4 Experiment Details and Additional Results	207
D.4.1 Online Function Approximation and Speed Benchmark	207
D.4.2 Trajectory Classification	207
E Appendix for Chapter 5	210
E.1 Proofs from Background	210
E.2 General Theory	211
E.3 Time-Varying Windows	218
E.3.1 Explanation of Time-Invariant LegS	218
E.4 Finite Windows	227
E.4.1 Derivation of LegT	227
E.4.2 Explanation of FouT	228
E.4.3 Function Approximation Error	233
E.4.4 Delay Network	234
E.5 Normalization and Timescales	243
F Appendix for Chapter 6	246
F.1 Statements and Proofs for Alternative SSM Structures	246
F.1.1 Recurrence Width	246
F.1.2 General HIPPO Operators have Low Recurrence Width	247

F.1.3	Efficient State Space Kernel for Quasiseparable Matrices	248
F.2	Proofs for DPLR SSMs	252
F.2.1	HIPPO Diagonalization	252
F.2.2	NPLR Representations of HIPPO Matrices	254
F.3	Proofs and Discussion of S4D Initializations	257
F.3.1	Proofs	257
F.3.2	General Low-rank Perturbations	259
G	Appendix for Chapter 7	261
G.1	General Sequence Modeling	261
G.1.1	General Training Protocol	262
G.1.2	Classification and Regression	263
G.1.3	Speech Commands Baselines	265
G.1.4	Time Series Forecasting	266
G.1.5	CIFAR Density Estimation	266
G.1.6	WikiText-103 Language Modeling	267
G.2	Additional Capabilities of S4	269
G.2.1	Efficient Inference: Autoregressive Generation Details	269
G.2.2	Efficient Training: Benchmarking Details	270
H	Appendix for Chapter 8	271
H.1	Additional Results	271
H.2	Experiment Details	272
H.2.1	Datasets	273
H.2.2	Models and Training Details	273
H.2.3	Automated Evaluations	276
H.2.4	Evaluation of Mean Opinion Scores	277
I	Appendix for Chapter 9	281
I.1	Theory Details	281
I.2	Experimental Details	284
I.2.1	Image Classification	284
I.2.2	Video Classification	284
I.2.3	Continuous Time Capabilities Experiments	285

List of Tables

2.1	Disambiguating the term SSM	35
3.1	Parameterization of Structured SSMs	50
3.2	Sequence model complexities	51
4.1	HIPPO function approximation error	66
4.2	HIPPO-LegS generalization to timescale change	69
5.1	Summary of time-invariant orthogonal SSMs	82
6.1	Summary of named S4 variants.	97
7.1	Pixel-level 1-D image classification	105
7.2	BIDMC Vital signs prediction	106
7.3	Speech Commands classification	107
7.4	Long Range Arena	108
7.5	Univariate long sequence time-series forecasting results	109
7.6	CIFAR-10 density estimation	111
7.7	WikiText-103 language modeling	112
7.8	S4 computation speed	114
7.9	Benchmarks vs. efficient Transformers	114
7.10	Ablation of S4 \mathbf{C} initialization	118
7.11	SSM parameterization ablations	119
7.12	Initialization and Trainability ablations	121
8.1	Summary of SaShiMi music and speech datasets	132
8.2	Results on AR modeling of Beethoven dataset	133
8.3	Effect of context length on Beethoven dataset	134

8.4	Negative log-likelihoods and MOS on YouTubeMix dataset	135
8.5	Architectural ablations and efficiency tradeoffs on YouTubeMix dataset	136
8.6	SC09 full results	137
8.7	SC09 diffusion model ablations	139
9.1	S4ND image classification	148
9.2	HMDB-51 Activity Recognition with ImageNet-pretrained models	150
9.3	Settings for continuous capabilities experiments	151
9.4	Zero-shot resolution change	151
9.5	Progressive resizing results	153
C.1	Sequence model complexities (extended)	186
G.1	Classification hyperparameters	263
G.2	BIDMC, SC, LRA hyperparameters	264
G.3	Univariate time-series forecasting (full results)	267
G.4	Multivariate time-series forecasting (full results)	268
I.1	Hyperparameters for image classification	285
I.2	Hyperparameters for video classification	286

List of Figures

1.1	Sequence models	2
2.1	Three representations of an SSM	21
3.1	Illustration of S4D	42
4.1	Illustration of the HIPPO framework	60
4.2	HIPPO reconstructions	66
4.3	Function approximation comparison between LegT and LegS	67
4.4	Discretization method comparison	68
5.1	New time-invariant HIPPO methods	72
5.2	Function reconstruction for prior HIPPO methods	75
5.3	HIPPO-LegT	76
5.4	Function reconstruction for new HIPPO methods	82
6.1	Visualization of diagonal approximation to LegS	94
6.2	Eigenvalues of S4D variants	95
7.1	Path-X visualizations	107
7.2	S4 vs Informer architectures	110
7.3	Properties of S4	112
7.4	Test-time sampling rate change	113
7.5	Timescale ablation on Path-X	117
7.6	HIPPO ablations with dense real SSM	123
7.7	HIPPO ablation with DPLR SSM	124
7.8	Reconstruction Task	126
7.9	Delay Task	127

8.1	SaShiMi architecture	129
8.2	Sample efficiency on SC09 dataset	136
8.3	Hurwitz stability	138
9.1	S4ND illustration	142
9.2	CIFAR-10 zero-shot comparison	151
9.3	Bandlimiting ablation	153
9.5	Effect of bandlimiting on learned kernels	154
F.1	Kernels for random low-rank perturbations	259
F.2	S4-FouT and diagonal approximation	260
H.1	Audio generation throughput	272
H.2	Audio generation throughput	272
H.3	YouTuBeMix MOS interface	277
H.4	SC09 MOS interface	278

Chapter 1

Introduction

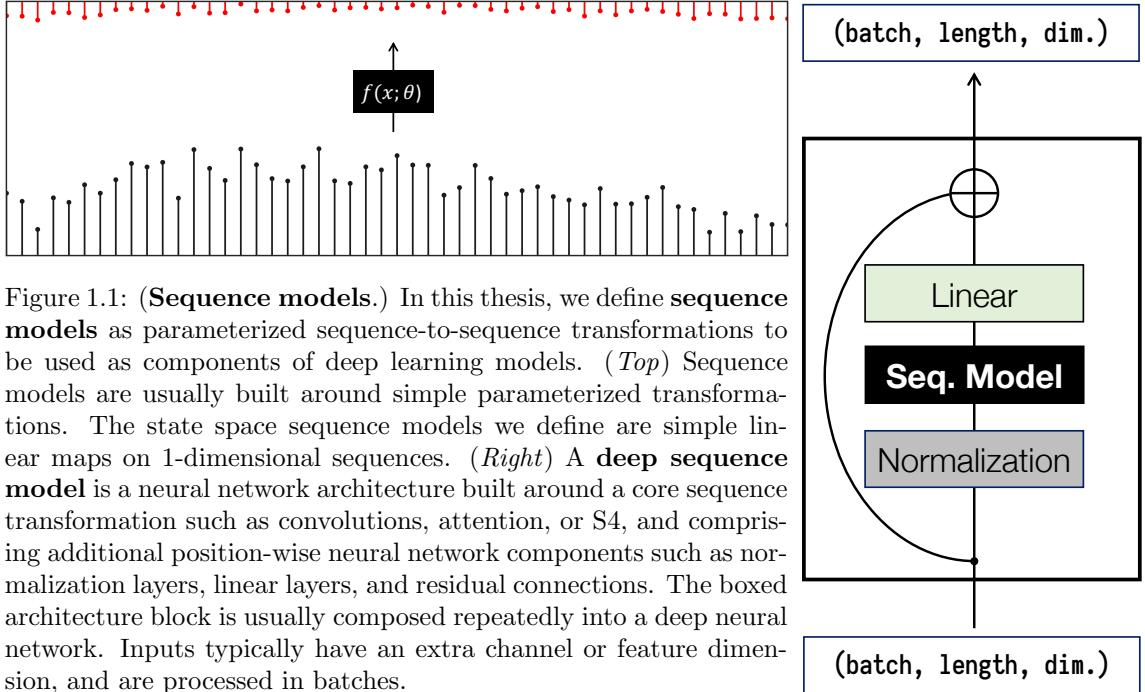
Deep learning methods have made significant advances in machine learning and artificial intelligence, achieving widespread success across scientific and industrial applications. A core class of models are sequence models, which are parameterized mappings operating on arbitrary sequences of inputs. These can be applied to a wide variety of complex sequential data processing tasks including natural language understanding, speech and audio, time-series analysis, and even indirect modalities that can be cast as sequences such as images [194, 148, 18, 94, 51].

1.1 Deep Sequence Models

Deep learning models for sequential data can be framed as sequence-to-sequence transformations built around simple mechanisms such as recurrence, convolutions, or attention [10].

These primitives can be incorporated into standard deep neural network architectures—forming the predominant **deep sequence model** families of recurrent neural networks (RNNs), convolutional neural networks (CNNs), and Transformers—which express powerful parameterized transformations that can be learned using standard deep learning techniques such as backpropagation with gradient descent. Figure 1.1 and Definition 1.1 illustrate the sequence model abstraction we use in this thesis, which is defined more formally with examples in Section 2.1.

Definition 1.1 (Informal). *We use **sequence model** to refer to a parameterized map on*



sequences $y = f_\theta(x)$ where inputs and outputs x, y are sequences of length L of feature vectors in \mathbb{R}^D , and θ are parameters to be learned through gradient descent.

Each of the above model families have facilitated tremendous successes for machine learning: for example, RNNs brought deep learning to machine translation [194], CNNs were the first neural audio generation model [148], and Transformers have completely revolutionized broad areas of NLP [217].

However, these models also come with tradeoffs inherited from their sequence mechanisms. For example, RNNs are a natural stateful model for sequential data that require only constant computation/storage per time step, but are slow to train and suffer from optimization difficulties (e.g., the “vanishing gradient problem” [156]), which empirically limits their ability to handle long sequences. CNNs specialize in local context, encode properties such as shift equivariance [33], and have fast, parallelizable training; but have more expensive sequential inference and an inherent limitation on the context length. Transformers have enjoyed tremendous success from their ability to handle long-range dependencies and their parallelizability, but suffer from a quadratic scaling in the sequence length. Another recent model family are the neural differential equations (NDEs) [25, 106], which are a principled mathematical model that can theoretically address continuous-time problems and long-term

dependencies, but are very inefficient [140].

These tradeoffs point to three broad challenges for deep sequence models at large.

Challenge 1: General-Purpose Capabilities

A broad goal of deep learning is to develop general-purpose building blocks that can be used across a wide range of problems. Sequence models provide a general framework for solving many of these problems with reduced specialization, as they can be applied to any setting that can be cast as a sequence – e.g. Vision Transformers for image classification with reduced 2-D information [51]. However, modern models typically still require substantial specialization to solve problems for specific tasks and domains, or to target particular capabilities. Examples of strengths of various model families include:

- RNNs: stateful settings that require rapid updating of a hidden state, such as online processing tasks and reinforcement learning.
- CNNs: modeling uniformly-sampled perceptual signals such as audio, images, and videos.
- Transformers: modeling dense, complex interactions in domains such as language.
- NDEs: handling atypical time-series settings such as missing or irregularly-sampled data.

Conversely, each model family can struggle with the capabilities it is not specialized for.

Challenge 2: Computational Efficiency

Applying deep sequence models in practice requires computing the functions they define (i.e. parameterized sequence-to-sequence map), which can take several forms. At training time, tasks can generally be formulated with loss functions over entire input sequences, where the central algorithmic concern is how to compute the forward pass efficiently. At inference time (deploying the model after it has been trained) the setting may change; for example, in *online processing* or *autoregressive generation* settings, inputs are only revealed one timestep at a time, and the model must be able to efficiently process these sequentially.

Both of these settings present challenges for different model families. For example, RNNs are inherently sequential and difficult to train on modern hardware accelerators such as GPUs and TPUs, which benefit from parallelizability. On the other hand, CNNs and Transformers have difficulty with efficient autoregressive inference because they are not

stateful; processing a single new input may scale with the entire context size of the model. More exotic models may come with additional capabilities (e.g. from the continuous nature of NDEs), but often makes them more difficult and slow to compute, (e.g. requiring calling expensive differential equation solvers [25]).

Challenge 3: Long-range Dependencies

Real-world sequential data can require reasoning over tens of thousands of time steps. Beyond the *computational* issues required to handle long inputs, addressing this also requires being able to *model* the complex interactions present in such long-range dependencies (LRD). In particular, difficulties can arise from being unable to capture the interactions in the data, such as if the model has a finite context window, or from optimization issues, such as the vanishing gradients problem when backpropagating through a long computation graph in recurrent models [156].

Long-range dependencies are a long-standing challenge for sequence models—due to these limitations in efficiency, expressivity, or trainability—and all standard model families such as NDEs, RNNs, CNNs, and Transformers include many specialized variants designed to address them. Modern examples include orthogonal and Lipschitz RNNs to combat vanishing gradients [5, 56], dilated convolutions to increase context size [11, 148], and an increasingly vast family of efficient attention variants that reduce their quadratic dependence on sequence length [104, 30, 203]. However, despite being designed for LRDs, these solutions still perform poorly on challenging benchmarks such as the Long Range Arena suite of tasks [202].

1.2 State Space Sequence Models

This thesis introduces a new family of deep sequence models based on linear state space models (SSMs). We define this SSM as a simple sequence model that maps a 1-dimensional function or sequence $u(t) \in \mathbb{R} \mapsto y(t) \in \mathbb{R}$ through an implicit latent state $x(t) \in \mathbb{R}^N$

$$\begin{aligned} x'(t) &= \mathbf{A}x(t) + \mathbf{B}u(t) \\ y(t) &= \mathbf{C}x(t) + \mathbf{D}u(t) \end{aligned} \quad (1.1)$$

SSMs are a foundational scientific model used in fields such as control theory [101], computational neuroscience [63], signal processing, and many more. The term SSM more broadly

refers to anything that models how latent variables evolve in a state space, and traditionally define probabilistic models of these dynamics (e.g. with stochastic transition matrices). Numerous flavors of these broader SSMs exist, which can vary the state space of x (e.g. continuous, discrete, or mixed spaces), observation space of y , transition dynamics, additional noise processes, or linearity of the system. SSMs historically often referred to variants of Hidden Markov Models (HMM) and linear dynamical systems (LDS), such as Hierarchical Dirichlet Processes (HDP-HMM) [204, 62, 98] and Switching Linear Dynamical Systems (SLDS) [21, 141, 65, 61, 125].

However, the state space model of equation (1.1)—which is continuous in both the state space and dynamics, and is completely linear and deterministic—has not been used as a deep sequence model in the sense of Definition 1.1. This thesis explores the many advantages of state space *sequence* models, and how they can be used to address the general sequence modeling challenges while overcoming their own limitations.

1.2.1 A General-Purpose Sequence Model

SSMs are a simple and fundamental model with many rich properties. They are closely related to model families such as NDEs, RNNs, and CNNs, and can in fact be written in several forms to acquire varied capabilities that normally require specialized models (Challenge 1).

- **SSMs are continuous.** The SSM itself is a differential equation. As such, it can perform unique applications of continuous-time models, such as simulating continuous processes, handling missing data [176], and adapting to different sampling rates.
- **SSMs are recurrent.** The SSM can be discretized into a linear recurrence using standard techniques, and simulated during inference as a stateful recurrent model with constant memory and computation per time step.
- **SSMs are convolutional.** SSMs are linear time-invariant systems known to be explicitly representable as a continuous convolution. Moreover, the discrete-time version can be parallelized during training using discrete convolutions, allowing for efficient training.

Consequently, SSMs excel as a general-purpose sequence model that can be efficient in both parallel and sequential settings and across a wide variety of domains (e.g. audio, vision, time-series). Chapter 2 presents background on SSMs and develops these properties of state space sequence models.

However, the generality of SSMs does come with tradeoffs. Naive SSMs still suffer from the other two challenges—perhaps even more so than other models—which has prevented their use as deep sequence models: (i) general SSMs are far slower than comparably-sized RNNs and CNNs, and (ii) they can struggle to remember long dependencies, e.g. inheriting the vanishing gradient problem from RNNs.

We address these challenges with new algorithms and theory for SSMs.

1.2.2 Efficient Computation with Structured SSMs (S4)

Unfortunately, general SSMs are infeasible to use in practice as deep sequence models because of prohibitive computation and memory requirements induced by the state representation $x(t) \in \mathbb{R}^N$ (Challenge 2).

For SSM state dimension N and sequence length L , computing the full latent state x alone requires $O(N^2L)$ operations and $O(NL)$ space – compared to a $\Omega(L + N)$ lower bound for computing the overall output. Thus for reasonably sized models (e.g. $N \approx 100$), an SSM uses orders of magnitude more memory than a comparably-sized RNN or CNN, rendering it computationally impractical as a general sequence modeling solution.

Overcoming this computational bottleneck requires imposing *structure* on the state matrix \mathbf{A} in a way that lends itself to efficient algorithms. We introduce families of such **structured state space sequence models (S4)** (or structured state spaces for short) with various forms of structured matrices \mathbf{A} , together with new algorithms that allow computing S4 models in any of its representations (e.g. recurrently or convolutionally) optimally efficiently.

Chapter 3 presents flavors of these efficient S4 models. The first structure uses a diagonal parameterization of the state matrix, which is extremely simple and general enough to represent almost all SSMs. We then generalize this by allowing a low-rank correction term, which is necessary to capture a special class of SSMs introduced later. By combining numerous technical ideas such as generating functions, linear algebraic transformations, and results from structured matrix multiplication, we develop algorithms for both of these structures with $\tilde{O}(N + L)$ time and $O(N + L)$ space complexity, which is essentially tight for sequence models.

1.2.3 Addressing Long-Range Dependencies with HIPPO

Independent of computational issues, a basic SSM still performs poorly empirically and suffers from being unable to model long-range dependencies (Challenge 3). Intuitively, one explanation is that linear first-order ODEs solve to an exponential function, and thus may suffer from gradients scaling exponentially in the sequence length. This can also be seen from their interpretation as a linear recurrence, which involves powering up a recurrent matrix repeatedly which is the cause of the well-known vanishing/exploding gradients problem for RNNs [156].

In [Chapter 4](#), we take a step back from SSMs and instead examine how to model LRDs with recurrent models from first principles. We develop a mathematical framework called HIPPO which formalizes and solves a problem that we call **online function approximation** (or memorization). This produces methods that aim to memorize a continuous function incrementally by maintaining a compression of its history. These resulting methods turn out to be specific forms of SSMs—despite being motivated completely independently!

[Chapter 5](#) refines this framework and connects it more rigorously to the SSM abstraction. It introduces a notion of **orthogonal SSM** that generalizes HIPPO and derives more instantiations and theoretical results, such as how to initialize all of the SSM parameters in a principled way.

Overview of HIPPO

Consider an input function $u(t)$, a fixed probability measure $\omega(t)$, and a sequence of N orthogonal basis functions such as polynomials. At every time t , the history of u before time t can be projected onto this basis, which yields a vector of coefficients $x(t) \in \mathbb{R}^N$ that represents an optimal approximation of the history of u with respect to the provided measure ω . The map taking the function $u(t) \in \mathbb{R}$ to coefficients $x(t) \in \mathbb{R}^N$ is called the **High-Order Polynomial Projection Operator (HIPPO)** with respect to the measure ω . In many cases, this turns out to have the form $x'(t) = \mathbf{A}x(t) + \mathbf{B}u(t)$ with closed-form formulas for (\mathbf{A}, \mathbf{B}) .

Combining HIPPO and S4

HIPPO provides a mathematical tool for constructing SSMs with important properties, while S4 is about computational representations. [Chapter 6](#) connects these formally and

shows they can be combined to acquire the best of both worlds. We show that the special matrices produced by HIPPO for addressing long-range dependencies can actually be written in the specific structured forms developed in Chapter 3. This provides concrete instantiations of S4 incorporating HIPPO, which results in a general sequence model that has rich capabilities, is very efficient, and excels on long-range reasoning.

1.2.4 Applications, Ablations, and Extensions

General Sequence Modeling Capabilities

Chapter 7 provides a comprehensive empirical validation of S4 methods across a wide variety of domains and tasks, where it advances the state-of-the-art for numerous benchmarks when incorporated into a generic simple deep neural network.

Particular highlights and capabilities include:

- *General sequence modeling.* With no architectural changes, S4 surpasses audio CNNs on speech classification, outperforms the specialized Informer model on time-series forecasting problems, and matches a 2-D ResNet on sequential CIFAR with over 90% accuracy.
- *Long-range dependencies.* On the LRA benchmark for efficient sequence models, S4 is as fast as all baselines while outperforming all Transformer variants by over 25% accuracy on average. S4 is the first model to solve the difficult LRA Path-X task (length-16384), achieving **96%** accuracy compared to 50% random guessing for all prior work.
- *Sampling resolution change.* Like specialized NDE methods, S4 can adapt to changes in time-series sampling frequency without retraining.
- *Large-scale generative modeling with fast autoregressive generation.* On CIFAR-10 density estimation, S4 is competitive with the best autoregressive models (2.85 bits per dim). On WikiText-103 language modeling, S4 substantially closes the gap to Transformers (within 0.5 perplexity), setting SOTA for attention-free models. Like RNNs, S4 can use its latent state to perform pixel / token generation 60× faster than standard autoregressive models on CIFAR-10 / WikiText-103.

Theory Ablations

Our treatment of S4 discusses many theoretical details of training SSMs, such as how to carefully initialize each parameter and how to incorporate the HIPPO framework. These are comprehensively empirically analyzed and ablated, validating all aspects of our SSM

theory. For example, we validate that HIPPO substantially increases the modeling power of SSMs, increasing performance by up to 15% on standard sequence model benchmarks compared to naive SSM instantiations. Algorithmically, our S4 algorithms improve over a naive SSM by orders of magnitude (e.g. 30 \times faster and 400 \times less memory usage).

Application: Audio Waveform Generation

As a sequence modeling primitive with many properties, S4 can be incorporated into different neural network architectures and used in multiple ways. **Chapter 8** presents an application of S4 to raw audio waveform generation, a challenging problem due to the high sampling rates of audio waveforms. It introduces the SASHIKI multi-scale architecture built around S4, which advances the state of the art in unconditional audio and speech generation, in multiple generative modeling paradigms including autoregression and diffusion. This application highlights the flexible capabilities of S4, including efficient training, fast autoregressive generation, and a strong inductive bias for modeling continuous signals.

Extension: Multi-dimensional Signals for Computer Vision

While we focus mainly on 1-dimensional sequences, some forms of data are inherently higher dimensional, such as images (2-D) and videos (3-D). The flexibility of sequence models can apply to these settings as well. **Chapter 9** presents S4ND, an extension of S4 from 1-D to multi-dimensional (N-D) signals. S4ND inherits the properties of S4 such as directly modeling the underlying continuous signal, with associated benefits such as better handling changes in input resolution, and is the first continuous model with competitive performance on larger vision tasks such as ImageNet.

1.3 Thesis Notes

An overarching theme in this line of work is the close interplay between *computational efficiency*, *modeling abilities*, and *empirical effectiveness*. Each of these cannot be advanced without considering the others, and the works that this thesis is based on were motivated and developed in a tightly woven manner that significantly deviates from the present presentation. This thesis has opted to separate these three factors as cleanly as possible into **Parts I to III** respectively, to better serve as a tutorial and reference on this subject matter.

We briefly note some alternate reading paths:

- Part I (Chapters 2 and 3) is a self-contained description of the S4 model family; for the practitioner, it contains all the material necessary to implement and use these models. Part II is not strictly necessary before applications in Part III.
- The first chapter of Part II (Chapter 4) is completely independent of Part I, and motivates SSMs from a different direction. This can be read as a standalone in any order, and historically was the progenitor of this line of work.

1.3.1 Bibliographic Remarks

This thesis is primarily based on the following previous publications.

- Chapter 2 builds on material from [74, 71, 75].
- Chapter 3 revises material from [71, 73].
- Chapter 4 revises [70].
- Chapter 5 revises [75].
- Chapter 6 extends material from [74, 71, 73] and includes material from [41].
- Chapter 7 revises material from [71, 75, 73].
- Chapter 8 revises [67].
- Chapter 9 revises [143].

[41] Christopher De Sa, Albert Gu, Rohan Puttagunta, Christopher Ré, Atri Rudra. “A Two-Pronged Progress in Structured Dense Matrix Vector Multiplication”. *Symposium on Discrete Algorithms (SODA)*. 2018.

[67] Karan Goel, Albert Gu, Chris Donahue, Christopher Ré. “It’s Raw! Audio Generation with State-Space Models”. *International Conference on Machine Learning (ICML)*. 2022.

[70] Albert Gu*, Tri Dao*, Stefano Ermon, Atri Rudra, Christopher Ré. “HIPPO: Recurrent Memory with Optimal Polynomial Projections”. *Advances in Neural Information Processing Systems (NeurIPS)*. 2020.

[71] Albert Gu, Karan Goel, Christopher Ré. “Efficiently Modeling Long Sequences with Structured State Spaces”. *International Conference on Learning Representations (ICLR)*. 2022.

- [73] Albert Gu, Ankit Gupta, Karan Goel, Christopher Ré. “On the Parameterization, Initialization of Diagonal State Space Models”. *Advances in Neural Information Processing Systems (NeurIPS)*. 2022.
- [74] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, Christopher Ré. “Combining Recurrent, Convolutional, and Continuous-time Models with the Linear State Space Layer”. *Advances in Neural Information Processing Systems (NeurIPS)*. 2021.
- [75] Albert Gu*, Isys Johnson*, Aman Timalsina, Atri Rudra, Christopher Ré. “How to Train Your HIPPO: State Space Models with Generalized Basis Projections”. *International Conference on Learning Representations (ICLR)*. 2023.
- [143] Eric Nguyen*, Karan Goel*, Albert Gu*, Gordon Downs, Preey Shah, Tri Dao, Stephen Baccus, Christopher Ré. “S4ND: Modeling Images and Videos as Multidimensional Signals with State Spaces”. *Advances in Neural Information Processing Systems (NeurIPS)*. 2022.

1.3.2 Notation and Conventions

Theorem styles

Theorems and Lemmas delineate major and minor results that we derive. We will use Propositions to denote important results from prior work outside of the material this thesis develops. We will frequently use Remarks to add additional connections and context, for example explaining how material relates between different chapters or has changed compared to the original papers, but they are generally not necessary to understand the flow of the main technical content.

Notation

We use bold capitals for important matrices, such as for model parameters (e.g. \mathbf{A}). Counting and indices are always 0-indexed; for example, $[n]$ denotes the set of natural numbers $\{0, 1, \dots, n - 1\}$. x' or \dot{x} is always reserved for derivatives of functions. e_n denotes the n -th standard basis vector $[0 \dots 1 \dots 0]$. Sequences are often denoted by (x_k) (e.g. $(x_k)_{k \in [L]}$ for a sequence of length L). Capital letters (e.g. L, M, N) are usually reserved for dimensions/axes. The real and imaginary part of complex numbers are denoted with a_{Re} and a_{Im} or $\Re(a)$ and $\Im(a)$.

Within prose, **technical terms** or definitions are frequently bolded. Italics are reserved for *emphasis*.

Part I

Structured State Spaces

Chapter 2

Sequence Modeling with State Space Models

This chapter defines state space models, introduces how they can be used as sequence models, and explores their theoretical properties and connections to other sequence models.

- Section 2.1 formalizes our notion of sequence models.
- Section 2.2 defines and provides necessary background on the classical state space model.
- Section 2.3 introduces state space models as a sequence model, focusing on various representations and ways to compute the model.
- Section 2.4 provides intuition and various interpretations of SSMs, particularly how they relate to other sequence models such as RNNs and CNNs.
- Section 2.5 concludes with a discussion on terminology for different types of SSMs, particularly how they relate to neural networks.

2.1 Background: The Sequence Model Framework

While “sequence model” is a generic and often overloaded term, we will use it to refer to a specific type of usage in deep learning.

Definition 2.1. We use the term **sequence model** to refer to a parameterized map on sequences $y = f_\theta(x)$ where $x \in \mathbb{R}^{L \times D}$ and $y \in \mathbb{R}^{L \times D'}$, and θ is an arbitrary collection of parameters. We will in general assume that $D = D'$, which is often referred to as the number of **channels** or **features**. L represents the input and output sequence length. Subscripts index into the first dimension, e.g. $x_k, y_k \in \mathbb{R}^D$.

Note that this definition of sequence model is still generic. For example, the composition of sequence models is a sequence model. We sometimes use terms to loosely distinguish between different flavors of “simple” versus “complex” sequence models.

Definition 2.2. **Sequence transformations** are “simple” parameterized maps (e.g. self-attention), including linear sequence models (e.g. 1-D convolutions). These are viewed as building blocks that do not define a full deep learning model by themselves; we will sometimes also call these sequence mechanisms. **Deep sequence models** are neural network architectures built around sequence transformations (e.g. Transformers, CNNs).

2.1.1 Learning with Deep Sequence Models

A deep sequence model typically comprises these components:

- An embedding or encoder layer $e_{in} : \mathbb{R}^{D_{in}} \rightarrow \mathbb{R}^D$ that operates position-wise on the sequence. Typical examples are an embedding lookup table for categorical inputs, or a linear projection parameterized by a weight matrix $\theta_{in} \in \mathbb{R}^{D_{in} \times D}$ for continuous inputs. (In this case, e_{in} is a matrix multiplication $e_{in}(x) = x\theta_{in}$.)
- A neural network block $f_\theta : \mathbb{R}^{L \times D} \rightarrow \mathbb{R}^{L \times D}$ built around a core sequence transformation. It is typically repeated K times with independent parameters $f_1 = f_{\theta_1}, \dots, f_K = f_{\theta_K}$.
- A projection or decoder layer $e_{out} : \mathbb{R}^D \rightarrow \mathbb{R}^{D_{out}}$ that operates position-wise, often a linear projection.
- A task-dependent loss function $\ell : \mathbb{R}^{D_{out}} \times \mathbb{R}^{D_{out}} \rightarrow \mathbb{R}$.

The dimensions D_{in} and D_{out} also depend on the task, usually based on the dimension of the inputs $x \in \mathbb{R}^{L \times D_{in}}$ and outputs $y \in \mathbb{R}^{L \times D_{out}}$ (or $y \in \mathbb{R}^{D_{out}}$)

The overall model is $f_\theta = e_{out} \circ f_K \circ \dots \circ f_1 \circ e_{in} : \mathbb{R}^{L \times D_{in}} \rightarrow \mathbb{R}^{L \times D_{out}}$ with parameters $\theta = (\theta_{in}, \theta_1, \dots, \theta_K, \theta_{out})$. Computing this function is often called computing the **forward pass** of the model.

The overall loss function on an input–output pair (x, y) is $L_\theta(x, y) = \ell(\hat{y}, y)$ where $\hat{y} = f_\theta(x)$. (Note that like the embedding/projection layers, the loss function can be broadcast over the sequence dimension L if the targets y are in $\mathbb{R}^{D_{out}}$ instead of $\mathbb{R}^{L \times D_{out}}$.)

Finally, the loss function on a data distribution $\mathcal{P}(x, y)$ is $L(\theta; \mathcal{P}) = \mathbb{E}_{(x, y) \sim \mathcal{P}} [L_\theta(x, y)]$. (For example, for a finite sized dataset, the loss is the average of $L_\theta(x, y)$ over all (x, y) pairs in the dataset.)

Learning a task (i.e. inferring model parameters from data) is simply viewing the data distribution \mathcal{P} as fixed and viewing $L(\theta; \mathcal{P})$ as a function in θ to minimize.

Many forms of learning exist in statistics and machine learning. In deep learning, models are almost always learned by some form of **backpropagation** that computes the gradients $\frac{\partial}{\partial \theta} L_\theta(x, y)$, combined with **gradient descent**. Moreover, backpropagation can usually be computed automatically by automatic differentiation frameworks for any sufficiently well-behaved model, in particular those that are composed of differentiable transformations. Consequently, in contrast to the many learning algorithms in machine learning at large, for our purposes the main *algorithmic* considerations of deep learning reduce entirely to computing the forward pass of the model efficiently.

Proposition 2.3 (Informal). Learning for deep sequence models reduces to computing its forward pass.

2.1.2 Example: Regression and Classification

We discuss some examples to show how the sequence model abstraction is used in deep learning for various tasks.

First, consider the case of a time-series regression task where the inputs are sequences $x \in \mathbb{R}^{L \times D_{in}}$ and targets are $y \in \mathbb{R}^{D_{out}}$. Here D_{in} is the number of time-series features (e.g. $D_{in} = 1$ for a univariate time-series), and D_{out} is the number of targets (e.g. $D_{out} = 1$ for predicting a single scalar target). The loss function is usually the mean squared error (MSE) $\ell(\hat{y}, y) = \|y - \hat{y}\|_2^2$.

In a typical sequence classification task, inputs are sequences $x \in \mathbb{R}^{L \times D_{in}}$ and targets are C-way classification labels $y \in [C]$. Here D_{out} is set to C , the targets are viewed as basis or “one-hot” vectors from $\{e_0, \dots, e_{C-1}\}$, and the loss function is the cross-entropy (CE) or negative log-likelihood (NLL) loss $\ell(\hat{y}, y) = -\sum y \log(\hat{y})$.

2.1.3 Example: Autoregressive Generative Modeling

A more involved and illustrative example is autoregressive generative modeling, which is the cornerstone of incredibly impactful deep learning models for generating data such as language (e.g. GPT [18]) and audio (e.g. WaveNet [148]).

Given a distribution over sequences $x = (x_0, \dots, x_{T-1})$, autoregressive generative models model the joint distribution as the factorized product of conditional probabilities

$$p_\theta(x) = \prod_{t=0}^{T-1} p_\theta(x_t | x_0, \dots, x_{t-1}).$$

Autoregressive models need two different modes or functionalities.

Training: First, the model must be trained on a distribution (dataset) of sequences: given a sequence of samples x_0, \dots, x_{L-1} , maximize the (log-)likelihood

$$\log p_\theta(x_0, \dots, x_{L-1}) = \sum_{k=0}^{L-1} \log p_\theta(x_k | x_0, \dots, x_{k-1})$$

where ℓ is the cross-entropy loss function.

Inference (Generation): After training, they can then be used to generate sequences from this distribution: given x_0, \dots, x_{k-1} as context, sample from the distribution represented by $p(x_k | x_0, \dots, x_{k-1})$ to produce the next sample x_k .

These two functionalities can be achieved exactly by a causal sequence model.

Definition 2.4. A *causal* sequence model f_θ is one in which y_k depends only on x_0, x_1, \dots, x_k .

Given a causal sequence model f_θ that maps $(x_0, \dots, x_{L-1}) \mapsto (\hat{y}_0, \dots, \hat{y}_{L-1})$, we let the targets y_k represent the conditional autoregressive data distribution $p(x_k | x_0, \dots, x_{k-1})$. (Here, y would have the same dimensionality as x , or $D_{out} = D_{in}$. For example, when modeling a sequence of categorical inputs over C classes, typically $x_k \in \mathbb{R}^C$ are embeddings of the classes and $y_k \in \mathbb{R}^C$ represents a categorical distribution over the classes.) Then the model's conditional distribution $p_\theta(x_k | x_0, \dots, x_{k-1})$ is represented by the vector $\hat{y}_k = f_\theta(x_0, \dots, x_{k-1})$. Maximizing the log-likelihood is the same as minimizing the standard cross-entropy loss by

the sequence shifted by one element,

$$\sum_{k=0}^{L-1} \log p_\theta(x_k|x_0, \dots, x_{k-1}) = \sum_{k=0}^{L-1} \ell(\hat{y}_{k-1}, x_k).$$

In other words, training an autoregressive generative model simply amounts to performing a forward pass of a sequence model, and using the cross-entropy loss against the same input sequence shifted by one element.

Similarly, generation from the trained model can be seen as a special computation mode for causal sequence models where only one input is revealed at a time.

Definition 2.5 (Step or inference mode of a causal sequence model). *One step of a causal sequence model with context length L is the map $x_L \mapsto y_L$. It can be viewed as the incremental cost of the forward pass $(x_0, \dots, x_L) \mapsto (y_0, \dots, y_L)$ assuming the model has already computed the previous forward pass $(x_0, \dots, x_{L-1}) \mapsto (y_0, \dots, y_{L-1})$. Note that because of causality, the outputs (y_0, \dots, y_{L-1}) do not change after the new input x_L is revealed.*

This can have a different computational complexity than the full forward pass: (i) it is only considering the cost of one output time-step y_L , and (ii) the computations involved in previous outputs can potentially speed this up. When we make a distinction between *training* and *inference* in discussions, we are usually referring to this definition.

2.2 Background: State Space Models

The continuous state space model (SSM) defines a linear mapping from an input signal (which is a function of time t) $u(t) \in \mathbb{R}^M$ to output signal $y(t) \in \mathbb{R}^M$ through a latent state $x(t) \in \mathbb{R}^N$:

$$x'(t) = \mathbf{A}(t)x(t) + \mathbf{B}(t)u(t) \quad (2.1a)$$

$$y(t) = \mathbf{C}(t)x(t) \quad (2.1b)$$

where $\mathbf{A}(t) \in \mathbb{R}^{N \times N}$, $\mathbf{B}(t) \in \mathbb{R}^{N \times M}$, $\mathbf{C}(t) \in \mathbb{R}^{M \times N}$. These are also commonly called the **state matrix**, **input matrix**, and **output matrix**. The domain of these functions is on $\mathbb{R}_+ := [0, \infty)$, i.e. (2.1a) defines an ODE with initial time 0.

Remark 2.1. *The second equation (2.1b) of the SSM is often defined as $y(t) = \mathbf{C}(t)x(t) + \mathbf{D}(t)u(t)$ where $\mathbf{D}(t) \in \mathbb{R}^{M \times M}$. This extra term involving \mathbf{D} can be viewed as a skip*

connection that does not interact with the state x , the most important part of the SSM (2.1a). Consequently, all of our technical results are independent of this \mathbf{D} term, with only one specific exception in Chapter 5. In this manuscript, we always leave out this term (or equivalently, assume $\mathbf{D} = 0$) unless explicitly mentioned.

Definition 2.6. Our treatment of SSMs will often consider (\mathbf{A}, \mathbf{B}) separately from \mathbf{C} . We will refer to an SSM as either the tuple (\mathbf{A}, \mathbf{B}) or $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ when the context is unambiguous. Formally, we may define an operator SSM such that (2.1a) can be written as $x = \text{SSM}(\mathbf{A}, \mathbf{B})(u)$ and (2.1b) can be written as $y = \text{SSM}(\mathbf{A}, \mathbf{B}, \mathbf{C})(u)$.

State space models are a **linear model**, in the sense that the output y is linear in the input u : $\text{SSM}(\mathbf{A}, \mathbf{B}, \mathbf{C})(u_1 + u_2) = \text{SSM}(\mathbf{A}, \mathbf{B}, \mathbf{C})(u_1) + \text{SSM}(\mathbf{A}, \mathbf{B}, \mathbf{C})(u_2)$

2.2.1 Linear Time Invariant (LTI) SSMs

SSMs can in general have dynamics that change over time, i.e. the matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, and vectors $\mathbf{B} \in \mathbb{R}^{N \times 1}$, $\mathbf{C} \in \mathbb{R}^{1 \times N}$ are a function of t in (2.1a) and (2.1b). However, when they are constant the dynamics are invariant through time; the SSM is then known as a **linear time-invariant (LTI)** system, and is equivalent to a (continuous) convolution (2.2).

Definition 2.7 (Convolution form of SSM). An LTI SSM (2.1) as a map $u(t) \mapsto y(t)$ can be expressed as

$$y(t) = (K * u)(t) = \int_0^\infty K(s) \cdot u(t - s) ds \quad \text{where} \quad K(t) = \mathbf{C} e^{t\mathbf{A}} \mathbf{B}. \quad (2.2)$$

The function $K(t)$ is called the **impulse response** which can also be defined as the output of the system when the input $u(t) = \delta(t)$ is the impulse or Dirac delta function. We will call this system an **LTI-SSM**, or **T-SSM** for short.

The majority of our treatment of SSMs will be in the LTI setting because of this connection to convolutions. In the remainder of this thesis, we will always use SSM to refer to a T-SSM and drop the T- for brevity, except in contexts where we wish to disambiguate different settings (Chapters 4 and 5). We will call systems that are not time-invariant **time-varying SSMs**. Notationally, the t -dependence is dropped for T-SSMs, i.e. $(\mathbf{A}(t), \mathbf{B}(t), \mathbf{C}(t))$ is replaced with $(\mathbf{A}, \mathbf{B}, \mathbf{C})$.

2.3 State Space Sequence Models

SSMs are broadly used in many scientific disciplines and have a long history starting from the 1960s with the Kalman filter [101]. However, our treatment of SSMs as a sequence model (culminating in S4) differs quite substantially in spirit, which is the focus of this section.

By Definition 2.1, we will define a state space *sequence* model as a sequence-to-sequence transformation from an input sequence $u \in \mathbb{R}^{L \times M}$ to an output sequence $y \in \mathbb{R}^{L \times M}$, parameterized by $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ and an additional parameter Δ (explained shortly). In this thesis, we will almost always use “SSM” to refer to this flavor of state space sequence model; we will sometimes call this an **SSSM** when we want to emphasize the distinction with traditional SSMs, for example when referring to the notion of forward passes, computation graphs, etc. At the end of this chapter, we will contrast different interpretations of SSMs in more detail (Section 2.5).

By Proposition 2.3, in order to *learn* a deep sequence models (i.e. update the parameters $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ by gradient descent), we only need to be able to *compute* the transformation (e.g. the map $u \mapsto y$) efficiently. A particularly nice feature of SSSMs is that they have many representations and modes of computation, each of which comes with distinct capabilities (Figure 2.1).

2.3.1 The Continuous Representation (Discretization)

Data in the real world is discrete instead of continuous, so equation (2.1a) must be discretized to be applied to an input sequence $u = (u_0, u_1, u_2, \dots)$ instead of continuous function $u(t)$. An additional **step size** parameter Δ is required that represents the resolution of the input. Conceptually, the inputs u_k can be viewed as uniformly-spaced samples from an implicit underlying continuous signal $u(t)$, where $u_k = u(k\Delta)$. We also call this a timescale, explained in Section 2.4.1.

Analogous to the fact that the SSM has equivalent forms either as a *differential equation* (2.1a) or a *continuous convolution* (2.2), the discrete-time SSM can be computed either as a *recurrence* or a *discrete convolution*.

To illustrate discretization, the simplest method is the well-known **Euler’s method** which turns the ODE $x'(t) = f(x(t))$ into the first-order approximation $x_k = x_{k-1} + \Delta f(x_{k-1})$.

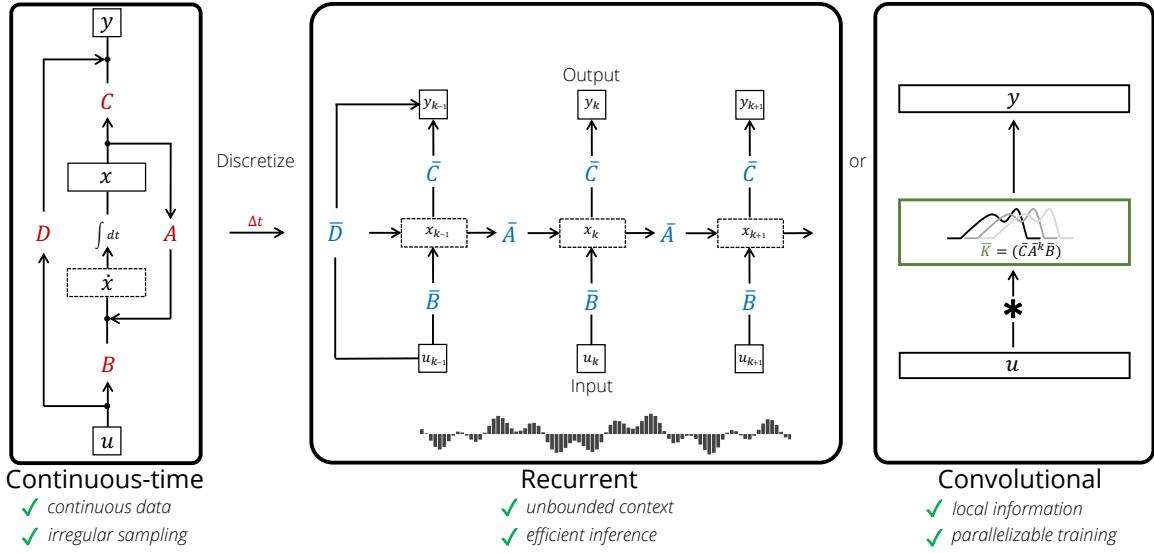


Figure 2.1: (**Three representations of an SSM.**) A state space sequence model is a map $(u_k) \in \mathbb{R}^L \rightarrow (y_t) \in \mathbb{R}^L$ defined by discretizing an SSM $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ with a timescale parameter Δ . (**Left**) As an implicit continuous model, an SSM acquires capabilities such as handling irregularly-sampled data by discretizing at different step sizes Δ . (**Center**) As a recurrent model, inference can be performed efficiently by computing the layer *time-wise* (i.e. one vertical slice at a time $(u_t, x_t, y_t), (u_{t+1}, x_{t+1}, y_{t+1}), \dots$), by unrolling the linear recurrence. (**Right**) As a convolutional model, training can be performed efficiently by computing the layer *depth-wise* in parallel (i.e. one horizontal slice at a time $(u_t), (y_t), \dots$), by convolving with a particular filter.

Applied to equation (1a), this is

$$\begin{aligned} x_k &= x_{k-1} + \Delta(\mathbf{A}x_{k-1} + \mathbf{B}u_k) \\ &= (\mathbf{I} + \Delta\mathbf{A})x_{k-1} + (\Delta\mathbf{B})u_k \\ &= \bar{\mathbf{A}}x_{k-1} + \bar{\mathbf{B}}u_k \end{aligned} \tag{2.3}$$

where $\bar{\mathbf{A}} := \mathbf{I} + \Delta\mathbf{A}$ and $\bar{\mathbf{B}} := \Delta\mathbf{B}$ are the *discretized* state parameters.

However, Euler's method can be unstable and more accurate methods such as the zero-order hold (ZOH) or bilinear transform (also called Tustin's method) [214] are commonly used. For our purposes, these simply provide alternate formulas for $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$, and are interchangeable (some tradeoffs are discussed in Chapter 3 and Appendix A.1).

(Bilinear)	(ZOH)
$\bar{\mathbf{A}} = (\mathbf{I} - \Delta\mathbf{A}/2)^{-1}(\mathbf{I} + \Delta\mathbf{A}/2)$	$\bar{\mathbf{A}} = \exp(\Delta\mathbf{A})$
$\bar{\mathbf{B}} = (\mathbf{I} - \Delta\mathbf{A}/2)^{-1} \cdot \Delta\mathbf{B}$	$\bar{\mathbf{B}} = (\Delta\mathbf{A})^{-1}(\exp(\Delta \cdot \mathbf{A}) - \mathbf{I}) \cdot \Delta\mathbf{B}.$

(2.4)

Appendix A.1 has a self-contained discussion of discretizations, including more discretization rules and simple derivations of all these methods.

Overall, we can think of discretization as the first step of the computation graph in the forward pass of an SS(S)M. It transforms the continuous parameters $(\Delta, \mathbf{A}, \mathbf{B})$ to discrete parameters $(\bar{\mathbf{A}}, \bar{\mathbf{B}})$ such that downstream computations only use $(\bar{\mathbf{A}}, \bar{\mathbf{B}})$. This transformation is defined by formulas $\bar{\mathbf{A}} = f_A(\Delta, \mathbf{A})$ and $\bar{\mathbf{B}} = f_B(\Delta, \mathbf{A}, \mathbf{B})$, where the pair (f_A, f_B) is a discretization rule such as the Euler, ZOH, or Bilinear (Tustin) methods defined above.

Notationally, we will often directly use $(\bar{\mathbf{A}}, \bar{\mathbf{B}})$ and suppress its dependence on the underlying parameters $(\Delta, \mathbf{A}, \mathbf{B})$ when it is clear.

2.3.2 The Recurrent Representation (Efficient Inference)

After discretizing (e.g. equation (2.3)), the discrete SSM is defined as

$$\begin{aligned} x_k &= \bar{\mathbf{A}}x_{k-1} + \bar{\mathbf{B}}u_k \\ y_k &= \mathbf{C}x_k \end{aligned} \tag{2.5}$$

(Note that discrete versions of parameters all have the same shapes as the original continuous counterparts: $\bar{\mathbf{A}} \in \mathbb{R}^{N \times N}$, $\bar{\mathbf{B}} \in \mathbb{R}^{N \times M}$, $u_k \in \mathbb{R}^M$, $x_k \in \mathbb{R}^N$, $y_k \in \mathbb{R}^M$.)

Equation (2.5) is now a *sequence-to-sequence* map $(u_k) \mapsto (y_k)$ instead of function-to-function $u(t) \mapsto y(t)$. Moreover the state equation is now a recurrence in x_k , allowing the discrete SSM to be unrolled one step at a time, in other words computed similar to any other recurrent model such as conventional RNNs. In the language of RNNs, $x_k \in \mathbb{R}^N$ can be viewed as a hidden state with transition matrix $\bar{\mathbf{A}}$.

Recurrent Mode Interpretation

The recurrent state $x_k \in \mathbb{R}^N$ carries the context of all inputs before time k . In other words, when unrolling the recurrence through equation (2.5), only the current state needs to be maintained. Therefore SSMs (and recurrent models in general) have efficient and stateful inference: unlike models such as CNNs and Transformers, they can consume a (potentially unbounded) sequence of inputs while only using *constant* computation and space per time step.

This is particularly useful for any online or autoregressive setting. A notable example is autoregressive generative models such as modern large language models, where generation

requires unrolling the model one step at a time.

2.3.3 The Convolutional Representation (Efficient Training)

The recurrent SSM (2.5) is not practical for training on modern hardware due to its sequentiality. Instead, there is a well-known connection between linear time-invariant (LTI) SSMs such as (2.1) and continuous convolutions. Correspondingly, (2.5) can actually be written as a discrete convolution.

Assume that the initial state is $x_{-1} = 0$. Because the discrete recurrence (2.5) is *linear*, it can be unrolled analytically, resulting in an equivalent convolutional form of the SSM.

$$\begin{aligned} x_0 &= \bar{\mathbf{B}}u_0 & x_1 &= \bar{\mathbf{A}}\bar{\mathbf{B}}u_0 + \bar{\mathbf{B}}u_1 & x_2 &= \bar{\mathbf{A}}^2\bar{\mathbf{B}}u_0 + \bar{\mathbf{A}}\bar{\mathbf{B}}u_1 + \bar{\mathbf{B}}u_2 & \dots \\ y_0 &= \mathbf{C}\bar{\mathbf{B}}u_0 & y_1 &= \mathbf{C}\bar{\mathbf{A}}\bar{\mathbf{B}}u_0 + \mathbf{C}\bar{\mathbf{B}}u_1 & y_2 &= \mathbf{C}\bar{\mathbf{A}}^2\bar{\mathbf{B}}u_0 + \mathbf{C}\bar{\mathbf{A}}\bar{\mathbf{B}}u_1 + \mathbf{C}\bar{\mathbf{B}}u_2 & \dots \end{aligned}$$

In other words, there is a simple closed form for the outputs y in terms of inputs u :

$$y_k = \mathbf{C}\bar{\mathbf{A}}^k\bar{\mathbf{B}}u_0 + \mathbf{C}\bar{\mathbf{A}}^{k-1}\bar{\mathbf{B}}u_1 + \dots + \mathbf{C}\bar{\mathbf{A}}\bar{\mathbf{B}}u_{k-1} + \mathbf{C}\bar{\mathbf{B}}u_k. \quad (2.6)$$

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

$$y = u * \bar{\mathbf{K}} \quad \text{where} \quad \bar{\mathbf{K}} = (\mathbf{C}\bar{\mathbf{B}}, \mathbf{C}\bar{\mathbf{A}}\bar{\mathbf{B}}, \dots, \mathbf{C}\bar{\mathbf{A}}^k\bar{\mathbf{B}}, \dots). \quad (2.7)$$

In other words, equation (2.7) is a single (non-circular) convolution. We call $\bar{\mathbf{K}}$ by several names, including the SSM (convolution) filter or kernel, or **state space kernel (SSK)** for short.

Computing (2.7) can be parallelized across the sequence length: instead of computing y_k from u_k one at a time, since a closed formula (2.6) exists, the entire output sequence y can be computed in parallel from the entire input sequence u .

Convolution Mode Interpretation

SSSMs can be interpreted very similarly to the linear convolution layers inside CNNs. The main distinction is that the state space kernel is actually *infinitely long*.

Remark 2.2. Note that the SSM kernel $\bar{\mathbf{K}}$ is actually infinite length, but is effectively limited to the length L of the input. It can be further truncated to any shorter length to act more as a traditional CNN (e.g. to emphasize locality, or for computational efficiency

reasons). We will frequently overload notation to use $\bar{\mathbf{K}}$ to directly refer to the truncated version to the input length L

$$\bar{\mathbf{K}} := \mathcal{K}_L(\bar{\mathbf{A}}, \bar{\mathbf{B}}, \mathbf{C}) := \left(\mathbf{C} \bar{\mathbf{A}}^k \bar{\mathbf{B}} \right)_{k \in [L]} = (\mathbf{C} \bar{\mathbf{B}}, \mathbf{C} \bar{\mathbf{A}} \bar{\mathbf{B}}, \dots, \mathbf{C} \bar{\mathbf{A}}^{L-1} \bar{\mathbf{B}}). \quad (2.8)$$

This leads to other departures compared to traditional CNNs. First, the convolution mode of an SSSM can be efficiently computed in an alternate way using three FFTs (assuming that the kernel $\bar{\mathbf{K}}$ is known). This is mathematically equivalent to a traditional “dense” convolution but has a different computation complexity depending on the kernel size.

Proposition 2.8 (FFT-convolution). *Let u, k be 1-D sequences of length L respectively. The circular convolution $u * k$ is equal to $\mathcal{F}^{-1}(\mathcal{F}(u) \circ \mathcal{F}(k))$, where \mathcal{F} and \mathcal{F}^{-1} are the Fourier Transform and Inverse Fourier Transform. This has computation complexity $O(L \log L)$.*

Proposition 2.9 (Causal convolution). *The causal (non-circular) convolution between u of length L and k of length K can be performed by padding each to length $L+K$ and performing a circular convolution.*

Corollary 2.10 (Convolution complexities). *For a sequence of length L and kernel of length K :*

- A traditional dense convolution has complexity $O(LK)$
- An FFT-convolution has complexity $O((L+K) \log(L+K))$ (with higher constant factor)

Second, the convolutional kernel is *implicit*; since it is infinitely long, it is generated from a smaller set of parameters, in this case $(\Delta, \mathbf{A}, \mathbf{B}, \mathbf{C}) \rightarrow \bar{\mathbf{K}}$ through (2.7). Many other methods of creating implicit kernels exist [174, 173, 124, 163], but generally lack the other (continuous and recurrent) properties of SSSMs.

Finally, we note that while computing the long convolution $y = u * \bar{\mathbf{K}}$ itself is fast through the standard FFT-convolution technique, generating the implicit kernel $\bar{\mathbf{K}}$ in (2.8) can be highly non-trivial and is the focus of Chapter 3.

Remark 2.3. *What we call the state space kernel is also known as the “Markov parameters” in the SSM literature. Our terminology underscores the distinction between SSMs and SSSMs; the latter are much more closely related to CNNs, whose parameters are called convolution filters or kernels.*

2.3.4 Summary of SSM Representations

Overall, the forward pass of an SS(S)M layer as a sequence model (Definition 2.1) consists of a discretization step, followed by different equivalent ways to compute the discrete SSM. We call the computation graph of first discretizing (2.1) and then computing equation (2.5) as “recurrent mode”. Similarly, we call the computation graph of first discretizing (2.1) and then computing equation (2.7) as “convolutional mode”.

Remark 2.4. *We sometimes also call these “RNN mode” and “CNN mode” for shorthand. Note however that an SSSM layer is not actually a neural network—just a linear sequence transformation (Definition 2.2)—and thus by itself (2.5) is not an RNN and (2.7) is not a CNN. See Section 2.5 for more discussion.*

Remark 2.5. *This section presented discretization from the recurrent view, and then derived the discrete convolution view by unrolling the recurrence. The discrete convolution view can also be derived by directly discretizing the continuous convolution view (2.2)*

$$y(t) = (K * u)(t) = \int_0^\infty \mathbf{C} e^{s\mathbf{A}} \mathbf{B} u(t-s) ds,$$

*which would involve approximating the integral in a way that preserves the convolutional structure of the model. (In this context, this is often called **quadrature**.)*

Remark 2.6. *As noted in Section 2.2.1, we only considered T-SSMs in this section, which is necessary for the equivalence between SSMs and convolutions. We note that generalizations of the other two modes are possible for general time-varying SSMs.*

2.3.5 A Note on SSM Dimensions

In Definition 2.1, we defined a sequence model as a map $\mathbb{R}^{L \times D} \rightarrow \mathbb{R}^{L \times D}$. On the other hand, the SSM (2.1) and consequently the SSSM (2.5) is defined as a map $\mathbb{R}^{L \times M} \rightarrow \mathbb{R}^{L \times M}$.

We view D as being the dimension of the input (independent of the model definition) and M as a hyperparameter defining the model. Our convention is that M must divide D , and applying an SSSM layer to an input with $D > M$ is done by chopping the model dimension D into pieces of size M , independently applying an SSM on each of these inputs (in $\mathbb{R}^{L \times M}$), and concatenating the outputs of the D/M SSMs back together at the end.

Note that this is exactly analogous to how multi-head attention works [217], so this can also be interpreted as a **multi-head SSM**, where $H = D/N$ is the number of heads.

Definition 2.11. Variants of SSMs with different model dimensions include:

- A **single-input single-output (SISO)** SSM is one in which $M = 1$.
- A **multi-input multi-output (MIMO)** SSM is one in which $M > 1$.
- We often use MIMO to specifically refer to the **maximal MIMO** case $M = D$.
- A **multi-head SSM** emphasizes the case when $M < D$, and we define $H = M/D$. Note that a SISO SSM is a multi-head SSM with a maximal number of heads $H = D$.

Definition 2.12 (SSM dimensions). Some terms for important dimensions include:

- D is the model size or dimension.
- N is the state size or dimension, or the “SSM state size” to disambiguate.
- $NH = ND/M$ is the “total state size”.

For example, a (maximal) MIMO SSM has total state size N whereas a SISO SSM has total state size ND .

When making comparisons of SSMs with other models, it can be useful to draw analogies to flavors of RNNs and CNNs.

Remark 2.7 (Relation to RNN dimensions). Traditionally, RNNs correspond to the MIMO ($M = D$ or $H = 1$) case. The total state dimension is N , which is an independent hyper-parameter from the model dimension D ; inputs are transformed between these dimensions via the input/output projection matrices $\bar{\mathbf{B}}$ and \mathbf{C} .

Many recent variants of RNNs have moved to a SISO flavor where the D channels are processed independently by a 1-D (scalar) recurrence. Notable examples include the cell state of LSTMs [86], the QRNN [17], SRU [120, 119], IndRNN [123], and RWKV [158]. In terms of dimensions, these models would correspond to an SSM with $M = 1$, $H = D$, and $N = 1$, for a total state size of D .

Remark 2.8 (Relation to CNN dimensions). For a MIMO SSM with $M = D$, then $C\bar{\mathbf{A}}^i\bar{\mathbf{B}} \in \mathbb{R}^{M \times M}$ and $\bar{\mathbf{K}}$ is a tensor of shape (L, M, M) . Convolving by $\bar{\mathbf{K}}$ can be interpreted in the standard CNN sense where these dimensions represent (sequence, out channels, in channels).

A multi-head SSM with $D|M$ is called a **grouped convolution** in the CNN literature [115].

When $M = 1$, or the input dimensions are independent channels that do not interact, this is also called a **depthwise** (or depthwise-separable) convolution in the CNN literature [195].

For the remainder of this thesis, we focus solely on the SISO case with $M = 1$ and $H = D$. The primary motivation is that the theoretical framework for online memorization (Part II) works with 1-dimensional inputs. However, we note that many popular and empirically effective models in the literature have also adopted this version of independent input channels, such as the RWKV RNN [158] and the ConvNeXt CNN [131]. We also note that most follow-up models to S4 preserve the SISO form, although some variants have appeared which switch to a MIMO formulation, notably S5 [192].

2.4 Interpreting SSM Representations

Section 2.3 focused on the computational aspects of SSSMs: how to implement them. In this section, we expand on the intuitions for SSSMs: how to interpret them compared to related sequence models. Note that this section is not necessary for using SSMs in practice, and in particular is not a prerequisite for Chapter 3.

- Section 2.4.1 discusses how to interpret the continuous nature of SSMs, especially the new Δ parameters.
- Sections 2.4.2 and 2.4.3 contrast SSSMs to RNNs and CNNs, focusing on their modeling capabilities and expressivity. In particular, we show how RNNs and CNNs can be captured by SSMs.

2.4.1 SSMs as Continuous Models

The continuous interpretation of SSSMs is a departure from standard sequence models. How should this model (and its extra Δ parameter) be interpreted, especially when the input u_k does not actually arise from sampling an underlying continuous signal $u(t)$?

Δ as a Modulating Parameter

We observe the following fact:

Lemma 2.13. *For all standard discretization methods (e.g. Euler, backward Euler, generalized bilinear transform, zero-order hold; see Appendix A.1), the discretized system depends on Δ and (\mathbf{A}, \mathbf{B}) only through their product $(\Delta\mathbf{A}, \Delta\mathbf{B})$.*

Corollary 2.14. *The SSSM $(\Delta, \mathbf{A}, \mathbf{B})$ (i.e. the SSM (\mathbf{A}, \mathbf{B}) discretized at step size Δ) is equivalent to the SSSM $(1, \Delta\mathbf{A}, \Delta\mathbf{B})$ (i.e. the SSM $(\Delta\mathbf{A}, \Delta\mathbf{B})$ discretized at step size 1): they compute the same sequence-to-sequence map.*

Thus instead of interpreting Δ as representing a fixed step size, we think of Δ as an extra parameter that simply modulates the core SSM parameters (\mathbf{A}, \mathbf{B}) .

Δ as a Timescale

The modulating interpretation of Δ means it can be viewed as controlling the length of dependencies in the (discrete) input. More precisely, it captures dependencies of length proportional to $\frac{1}{\Delta}$. One intuition for this follows from noting that scaling the linear ODE (2.1a) preserves the same dynamics but simply changes the rate of evolution of the system.

Lemma 2.15. *The ODE $y' = c\mathbf{A}y + c\mathbf{B}u$ evolves at a rate c times as fast as the SSM $x' = \mathbf{Ax} + \mathbf{Bu}$, in the sense that the former maps $u(ct) \mapsto x(ct)$ if the latter maps $u(t) \mapsto x(t)$.*

For this reason, we also refer to Δ as a **timescale**. For important new classes of SSMs that we introduce, this connection can be formalized much more precisely (Chapter 5).

Adjusting Sampling Rates with Δ

Although Δ is not considered a step size of the data, it can still be interpreted as a *relative* quantity with respect to the true step size. In particular, if the step size of the data is scaled by c (e.g. the sampling rate of a time-series is scaled by $1/c$), then the model's Δ parameter should also be scaled by c .

As a concrete example, consider an input sequence $x^{(1)}$ and another sequence $x^{(2)}$ which duplicates every element. In other words, $x^{(2)}$ can be viewed as generated from the same data as $x^{(1)}$ but at twice the sampling rate (or half the step size). Then it can be verified that $\text{SSM}(\Delta, \mathbf{A}, \mathbf{B}, \mathbf{C})(x^{(1)}) = \text{SSM}(\Delta/2, \mathbf{A}, \mathbf{B}, \mathbf{C})(x^{(2)})$ (up to discretization error; the equality is exact for ZOH).

Δ in Other Continuous Models

Other continuous-time models exist in deep learning; in fact, the first RNNs were often interpreted as approximations of continuous dynamical systems. Discretization with a step size Δ is often an intrinsic part of these models. However, most NDEs and ODE-based RNN models have it as a critical but non-trainable *hyperparameter* [218, 178], while SSSMs

can *train* it to control its lengths of dependencies.

Compared to classical SSMs (Section 2.5), we emphasize that the following is most important intuition from this section.

Remark 2.9. *In classical SSMs, Δ is interpreted as a step size which is an intrinsic property of the data. By contrast, in SSSMs Δ is interpreted as a trainable parameter that represents timescales.*

Δ in RNNs and CNNs

Finally, we remark that since the recurrent and convolutional representation of SSMs is downstream of discretization (Section 2.3.4), Δ has additional interpretations when viewing an SSSM directly as an RNN or CNN.

In the next subsection, we show that the gating mechanism of classical RNNs is a version of learning Δ . Finally, as a CNN, the timescale Δ can be viewed as controlling the “width” of the convolution kernel.

2.4.2 SSMs as Recurrences

Section 2.3.2 discussed how to interpret and compute an SS(S)M as a linear recurrence. However, while an SSM is a linear recurrence, popular RNN models are *nonlinear* sequence models with activation functions between each time step. This raises the question of how expressive SSMs are compared to general RNNs.

Gating Mechanism of RNNs

We show two results relating RNNs and ODEs that may be of broader interest. Our first result says that the ubiquitous **gating mechanism** of RNNs can actually be interpreted as the analog of a step size or timescale Δ .

Lemma 2.16. *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 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 Δ . The simplest way to parameterize a positive function is via the exponential function $\Delta = \exp(z)$ applied to any expression z . Substituting this into the Backwards-Euler discretization (Appendix A.1) exactly produces the gated recurrence. \square

We emphasize that the gating mechanism, which is empirically critical for RNN performance, is commonly perceived as a *heuristic* to smooth optimization [86]. In contrast, Lemma 2.16 provides an elegant interpretation and says that it could actually have been derived from first principles – even up to the exact sigmoid formula!

Picard Iteration as Depth-wise Approximation of Nonlinear Dynamics

While Lemma 2.16 involves approximating continuous systems using discretization, the second result is about approximating them using Picard iteration, a classical technique for ODEs (A brief description is provided in Appendix A.1). Roughly speaking, each layer of a deep *linear* RNN can be viewed as successive Picard iterates $x_0(t), x_1(t), \dots$ approximating a function $x(t)$ defined by a *nonlinear* ODE. This shows that we do not lose modeling power by using linear instead of nonlinear recurrences, and that these nonlinearities in the “time” dimension can instead be moved to the “depth” direction of deep neural networks without sacrificing expressivity.

Lemma 2.17. *(Infinitely) deep stacked SSM layers of order $N = 1$ with position-wise nonlinear functions can (locally) approximate the nonlinear ODE $\dot{x}(t) = -x + f(t, x(t))$.*

Koopman Operators

Although beyond the scope of this thesis, we mention a third connection between linear and nonlinear dynamical systems through the lens of Koopman operator theory. Orvieto et al. [150] discuss how interleaving linear RNN layers with nonlinear “feed-forward” blocks is sufficient to approximate highly nonlinear systems (under sufficient regularity conditions).

Discussion: RNNs as SSMs

Lemmas 2.16 and 2.17 suggest that general RNNs can be represented as deep neural networks built around (linear) SSMs. While we believe that these results are conceptually solid and give substantial insights into the RNN–SSM connection, they also have limitations.

- The way that current SSSMs (such as S4) treat Δ is somewhat restricted compared to gated RNNs: T-SSMs define Δ as a data-independent parameter that is constant for each time-step, while the gates of RNNs are often functions of the input u that can vary across time-steps. (Note that this is not a fundamental limitation, and it

would be straightforward to define time-varying SSSMs computed in recurrent mode whose Δ parameter can be functions of the data.)

- The Picard and Koopman connections of linear to nonlinear RNNs are more conceptual, for example applying in the limit of infinite depth or width; concrete quantitative or non-asymptotic analysis of the dynamics would require more work.

Nevertheless, we emphasize that these conceptual connections are supported by substantial empirical evidence that deep linear SSMs/RNNs perform better than classical RNNs. Beyond the strong empirical results for S4 developed in this thesis, we note that many of the most popular and effective RNN variants developed in a similar direction. These RNNs such as the LSTM [86], GRU [31], QRNN [17], and SRU [120], involve a hidden state $x_t \in \mathbb{R}^H$ that involves independently “gating” the H channels. Thus by Lemma 2.16, they actually also approximate an ODE of the form in Lemma 2.17. Overall, SSMs and these popular RNN models can be seen to all approximate the same type of underlying continuous dynamics, by using nonlinear approximations in the depth direction and discretization (gates) in the time direction.

Appendix B.1 gives more precise statements and proofs of these results.

2.4.3 SSMs as Convolutions

As in Section 2.4.2, a similar question can be raised about the expressivity of SSSMs compared to traditional CNNs. For a model to be both recurrent and convolutional, one might expect it to be limited in other ways. In particular, the convolutional representation was derived from unrolling a particular LTI recurrence (Section 2.3.3); this suggests that only restricted families of convolutions can be represented by an SSM.¹ However, classical results imply that essentially all convolutions can in fact be expressed as an SSM.

Convolutions are LTI-SSMs

Recall that equation (2.2) says that any (continuous, linear time-invariant) SSM can be written as a convolution by the impulse response of the system K :

$$\text{SSM}(\mathbf{A}, \mathbf{B}, \mathbf{C})(u) = K * u \quad \text{where } K(t) = \mathbf{C}e^{t\mathbf{A}}\mathbf{B}$$

¹Similar observations were made in [28, 174].

Conversely, any convolutional filter $K(t)$ that is a rational function of degree N can be represented by a state space model of dimension N [224]. Thus, an arbitrary convolutional filter can be approximated by a rational function (e.g. by Padé approximants [13]) and represented by an SSM. In other words, (continuous) SSMs can represent any (continuous) convolutional model as the state size $N \rightarrow \infty$. (In fact, the continuous constraint is not fundamental; finite discontinuities in the convolution kernel are also representable, which we construct explicitly in Chapter 5.)

Interpreting SSMs as Convolutional Systems

Definition 2.18 provides a very useful way of interpreting the convolutional kernel of SSMs.

Definition 2.18. *Given a T-SSM (\mathbf{A}, \mathbf{B}) , $e^{t\mathbf{A}}\mathbf{B}$ is a vector of N functions which we call the **SSM basis**. The individual basis functions are denoted*

$$K_n(t) = \mathbf{e}_n^\top e^{t\mathbf{A}}\mathbf{B} \quad (2.9)$$

, which satisfy

$$x_n(t) = (u * K_n)(t) = \int_{-\infty}^t K_n(t-s)u(s) ds. \quad (2.10)$$

This definition is motivated by noting that the SSM convolutional kernel is a linear combination of the SSM basis controlled by the vector of coefficients \mathbf{C} : $K(t) = \sum_{n=0}^{N-1} \mathbf{C}_n K_n(t)$.

Remark 2.10. *It may be useful as a thought experiment to motivate SSMs from first principles from the final goal of representing infinitely-long convolutions. Viewing a convolution kernel $K(t)$ as an arbitrary (continuous) function $[0, \infty) \rightarrow \mathbb{R}$, the central question is how to represent this function in a compressed form. One approach is to parameterize it with a neural network, which was taken by parallel work [174]. Another intuitive approach is to view $K(t)$ as a linear combination of basis functions, e.g. in the spirit of Fourier analysis. This approach leads to Definition 2.18.*

Finally, as in Section 2.4.2 which contrasted deep neural networks built on SSMs against RNNs, we can compare them against CNNs. This analogy is very tight because a convolution layer is a simple linear transformation, just like an SSM; an overall neural network is made nonlinear by activations through the depth of the network. Thus a deep SSM model is exactly a depthwise-separable CNN (Remark 2.8) but with global, implicit convolution kernels.

2.5 Discussion: Naming Conventions for SSMs

With the development of SSMs for deep sequence modeling, the umbrella term “SSM” has become overloaded with many meanings. We discuss and introduce terminology to disambiguate these usages when necessary.

2.5.1 Classical State Space Models

We use the term **classical SSM** or **statistical SSM** to refer to older flavors of SSMs such as the Kalman filter commonly used in statistics and engineering disciplines. These are typically *statistical models* with characteristics such as:

1. They define *probabilistic models* of a data generating process.
2. Their parameters are learned through standard tools of *statistical inference*.
3. In order to be tractable, models are typically simple; for example a single instance of equation (2.1) often defines a complete model.

Many other details of classical SSMs often diverge from our usage. For example, while we view u as fixed inputs and $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ as learnable parameters, in control theory their role is often swapped so that the system dynamics are fixed and u is the object to learn or “control”. Similarly, a common objective of statistical inference for SSMs is to infer the state x itself based on the sequence observations, while SSSMs do not care about the state per se and frequently do not even materialize it (as in the convolutional mode).

2.5.2 State Space Sequence Models

In contrast to statistical SSMs, the SSMs defined in this Chapter differ in several ways because of their intended use as sequence models in deep learning (Section 2.1).

1. They define *deterministic maps* of sequences (Definition 2.1), which can represent feature transformations or feature extraction.
2. Their parameters are learned through *gradient descent*, and algorithmically the main challenge is computing the forward pass (Proposition 2.3).
3. They are useful only when composed multiple times and incorporated into nonlinear deep neural network architectures (e.g. as discussed in Section 2.4.2); an individual layer by itself is not a model.

Remark 2.11. As discussed in Chapter 1, and as the focus of Chapter 3, using these models also necessarily requires imposing additional structure on the SSM to be computationally feasible for deep neural networks. Thus all practical SSSMs are actually structured SSSMs, and S4 or “structured SSM” can also be used as umbrella terms for SSSM in practice.

We find it more intuitive to view SSSMs as analogous to the convolution layers of CNNs, rather than being derived from classical SSMs. For example, convolution layers are often interpreted as performing automatic feature extraction, which was partly the original motivation of CNNs. Similarly, an individual S4 layer can be interpreted as deterministically extracting a specific interpretable feature (Part II). This stands in contrast to the typical probabilistic model interpretation of classical SSMs.

Another important departure from classical SSMs is the additional parameter Δ , which has a very different interpretation (Section 2.4.1).

2.5.3 Deep State Space Models

In the literature, SSSMs such as S4 have also started being called deep SSMs because of their intended use as a deep sequence model. We instead reserve the **deep SSM** term to refer to a *deep neural network architecture* built around SSSMs, just as CNNs are neural network architectures built around linear convolution layers. In particular, deep SSMs are architectures combining SSSM/S4 layers with other common neural network components such as residual connections [81], normalization [92, 7], linear projections and nonlinear activations, and so on. Examples that have appeared in the literature include GSS [136], H3 [37] and Mega [134].

We will also call deep SSMs **state space neural networks (SSNN)** as a convenient shorthand that underscores the analogy to RNNs and CNNs. Note that the terms S4 and SSNN draw the same distinction between sequence transformations and deep sequence models described in Definition 2.2.

Summary

Despite sharing the same basic form (2.1), the flavor of SSMs introduced by S4 are much more closely related to modern neural networks than classical SSMs. While we will often just use the simpler umbrella term “SSM” for convenience when context is clear, we believe it is important to distinguish between the meanings of this term when appropriate.

We summarize this categorization of terms in Table 2.1, and conclude with analogies to

Table 2.1: Various meanings of the umbrella term SSM, and more specific disambiguations.

Names	Meaning	Examples
classical SSM	Variants of traditional SSMs	Kalman filter [101], HMMs
statistical SSM	used for statistical modeling	
SSSM S4	Linear (structured) SSM layers used as a deep sequence model	S4, S5 [192], DSS [76]
deep SSM SSNN	Deep neural network architectures built around linear S4 layers	GSS [136], H3 [37], Mega [134]

other neural network families to underscore their meaning:

- *Transformers* are deep neural networks built around *attention* layers.
- *CNNs* are deep neural networks built around *linear convolution* layers.
- *SSNNs* are deep neural networks built around *linear SSSM/S4* layers.

Chapter 3

Computing Structured SSMs

Chapter 2 introduced state space sequence models and their rich properties, such as being able to be expressed through multiple representations. However, these theoretical properties can also make them more difficult to work with in practice. For example, being an implicit continuous model affords flexibility, but means that computing an SSM as a convolution requires first transforming the parameters which can be very expensive.

This chapter fills in remaining details of how to define and compute S4 models.

- Section 3.1 formalizes the computational challenges of computing SSMs as a sequence model, and motivates the need for structured SSMs (S4). The next two sections then present algorithms for two classes of S4.
- First, Section 3.2 describes a very structured class of diagonal SSMs (S4-Diag) which are very simple to compute, and are sufficiently expressive to represent most (but not all) SSMs.
- Section 3.3 generalizes diagonal SSMs to a class S4-DPLR which is necessary to represent special SSMs that are the subject of Part II, and outlines a more sophisticated algorithm to compute its SSM kernel.
- Section 3.4 describes several more details in the parameterization of these S4 methods.
- Section 3.5 contrasts S4-Diag and S4-DPLR in more depth.
- Finally, Section 3.6 compares the computational complexity of S4 with other sequence models, and briefly surveys other types of structured SSMs that have appeared in the

literature.

Remark 3.1. As the more general structure (and for historical reasons), in this chapter and the rest of this thesis, the term $S4$ without qualifiers usually refers to $S4$ -DPLR for short. Similarly, $S4D$ is shorthand for $S4$ -Diag.

3.1 Motivation: Computational Difficulty of SSMs

We begin by formally defining SSMs that are optimally efficient.

Definition 3.1 (Recurrent mode computation). *Unrolling one step of the SSM recurrence (2.5) requires **matrix-vector multiplication (MVM)** by the discretized state matrix $\bar{\mathbf{A}}$. An SSM has optimal recurrent mode if this MVM can be computed in $O(N)$ time (or $\tilde{O}(N)$) instead of $O(N^2)$.*

Definition 3.2 (Convolution mode computation). *The SSM convolution mode requires materializing the **state space kernel $\bar{\mathbf{K}}$** (2.8). An SSM has optimal convolution mode if this SSK can be computed in $O(L + N)$ time (or $\tilde{O}(L + N)$).*

Problem 1. For a given SSM structure, find an algorithm that satisfies Definition 3.1.

Problem 2. For a given SSM structure, find an algorithm that satisfies Definition 3.2.

Note that the complexities in Definitions 3.1 and 3.2 are optimal in the sense that they are lower bounds for these operations: it requires $O(N)$ time to read the SSM parameters ($\mathbf{A}, \mathbf{B}, \mathbf{C}$) and $O(L)$ time to write the kernel $\bar{\mathbf{K}}$.

Remark 3.2. Definition 3.2 is really about the complexities needed to compute the sequence transformation $(u_k)_{[L]} \mapsto (y_k)_{[L]}$ (e.g. for training a deep sequence model), which is not necessarily tied to the convolutional representation.

However, note that the recurrent mode (2.5) requires materializing the recurrent state (x_k) , which by itself would take $O(LN)$ time and space across the whole sequence. Thus the recurrent mode cannot be used to achieve an efficient (SISO) SSM, which should have $O(L + N)$ time and space. This is one motivation for needing an alternative representation, and Definition 3.2 is phrased directly for convolution mode.

Remark 3.3. If the SSM parameters are fixed (e.g. at inference time after training), Problem 2 is avoided entirely because the SSM kernel $\bar{\mathbf{K}}$ can be cached. However, the kernel needs to be repeatedly computed during training time.

Additionally, Problem 1 is not free even at inference time, where it is often more important (e.g. for autoregressive generation).

In the remainder of this section, Sections 3.1.1 and 3.1.2 have a lengthier discussion of the difficulty of Problem 1 and Problem 2.

3.1.1 Discussion: General Recurrent Computation

Computing $\bar{\mathbf{A}}$ can be expensive, but it can usually be cached once per sequence, and then each step requires the time for MVM by $\bar{\mathbf{A}}$. However, MVM by a general $N \times N$ matrix requires $O(N^2)$ operations. Many families of structured matrices \mathbf{A} exist with fast MVM; but even then, it is not always true that $\bar{\mathbf{A}}$ does. For example, using the ZOH discretization requires MVM by the matrix exponential $\exp(\mathbf{A})$ – which is both itself difficult to compute in general, and also might not have fast MVM (depending on the exact structure of \mathbf{A}).

Similarly, if using the bilinear discretization instead (equation (2.4)), this requires a separate MVM by \mathbf{A} as well as MVM by a term related to the inverse of \mathbf{A} , $(\mathbf{I} + \Delta\mathbf{A})^{-1}$. It is not always guaranteed that the latter has fast MVM even when the former does.

Remark 3.4. *In general, unless \mathbf{A} is extremely structured (e.g. diagonal matrices in Section 3.2), the bilinear discretization will more easily lend itself to fast algorithms because working with matrix inverses is easier than with matrix exponentials.*

3.1.2 Discussion: General Convolutional Computation

It is easy to see that naively computing (2.8) for a general matrix \mathbf{A} takes time LN^2 . Trying to simplify the computation faces a difficult tradeoff between time, space, and depth¹. To really illustrate the difficulty of computing the SSM kernel, let us consider some potential simplifications and alternate computation strategies.

Structured \mathbf{A} . Suppose that \mathbf{A} is highly structured and $\bar{\mathbf{A}}$ has linear time MVM. Unrolling (2.8) still uses $O(LN)$ operations to compute

$$(\bar{\mathbf{B}}, \bar{\mathbf{AB}}, \bar{\mathbf{A}}^2\bar{\mathbf{B}}, \dots, \bar{\mathbf{A}}^{L-1}\bar{\mathbf{B}}) \in \mathbb{R}^{N \times L} \quad (3.1)$$

by repeatedly applying the MVM. Moreover, this is sequential with $O(L)$ depth, negating the parallelizable motivation of the convolutional mode and becoming essentially equivalent

¹Also called *span*, the number of sequential dependencies required in the computation graph, which is a measure of parallelizability.

to the recurrent mode.

Smaller depth. Equation (3.1) can be computed in a parallelizable manner by the squaring technique for exponentiation, i.e. batch multiplication by $\bar{\mathbf{A}}, \bar{\mathbf{A}}^2, \bar{\mathbf{A}}^4$, etc. However this increases the time back to N^2L (or $N^{\omega-1}L$) unless all powers of $\bar{\mathbf{A}}$ are structured; worse, this requires materializing the entire tensor (3.1) with $O(NL)$ space.

Frozen parameters. Consider the much simpler case where Δ, \mathbf{A} , and \mathbf{B} are all fixed (non-trainable) parameters, therefore allowing some partial computations to be cached, in particular the most expensive one (3.1) (Note that it does not make sense to fix \mathbf{C} as well, as then the entire convolution kernel is non-trainable). But this still requires $O(NL)$ time and space (compared to the ideal bound $O(L)$ for both).

In practice, the memory required to cache this large matrix is the limiting factor [74], and of course performance also suffers in this restricted setting where many of the important parameters are not trainable.

Numerical stability. Even when the above problems can be mitigated for special structures for \mathbf{A} , other problems may arise. For example, early versions of the S4 model found an algorithm that addressed these issues in exact arithmetic, but was numerically unstable in floating point (Section 6.1).

In fact, the repeated squaring version was found to be best in practice [74], which uses $O(LN^2)$ time and $O(LN)$ space – far higher than the bounds in Definition 3.2. For moderate values of N (empirically, N is often chosen to be in the range 16 – 256), this is very slow and has infeasible memory requirements when used in a modern deep neural network.

3.1.3 Structured SSMs

The difficulty of computing general SS(S)Ms is fundamental, and overcoming it requires imposing *structure* on the SSM that allows computing it in a fast, memory efficient, and numerically stable way. The remainder of this chapter presents various parameterizations of **structured state space sequence models** (S4) that satisfy the optimal efficiency bounds of Definitions 3.1 and 3.2.

Remark 3.5. We use *structured SSM* to refer to an SSM with a structured \mathbf{A} matrix. In turn, a *structured (square) matrix* is one that can be represented in fewer than quadratic

parameters through a compressed representation, and has fast algorithms (such as computing MVM) by operating directly on this compressed representation.

3.2 Diagonally Structured State Space Models

To overcome the SSM computation bottleneck, we use a structural result that allows us to transform and simplify SSMs.

Lemma 3.3. *Conjugation is an equivalence relation on SSMs:*

$$(\mathbf{A}, \mathbf{B}, \mathbf{C}) \sim (\mathbf{V}^{-1}\mathbf{A}\mathbf{V}, \mathbf{V}^{-1}\mathbf{B}, \mathbf{C}\mathbf{V}).$$

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

$$\begin{array}{ll} x' = \mathbf{A}x + \mathbf{B}u & \tilde{x}' = \mathbf{V}^{-1}\mathbf{A}\mathbf{V}\tilde{x} + \mathbf{V}^{-1}\mathbf{B}u \\ y = \mathbf{C}x & y = \mathbf{C}\mathbf{V}\tilde{x} \end{array}$$

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

Lemma 3.3 says that the state space $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ is exactly equivalent to the state space $(\mathbf{V}^{-1}\mathbf{A}\mathbf{V}, \mathbf{V}^{-1}\mathbf{B}, \mathbf{C}\mathbf{V})$; in other words, they express the exact same map $u \mapsto y$. This is also known in the SSM literature as a **state space transformation**.

Lemma 3.3 motivates putting \mathbf{A} into a canonical form by conjugation, which is ideally more structured and allows faster computation.²

A very natural choice for such a form are diagonal matrices, which are perhaps the most common canonical form. It is well-known that almost all matrices diagonalize over the complex plane.

Proposition 3.4. *The set $\mathcal{D} \subset \mathbb{C}^{N \times N}$ of diagonalizable matrices is dense in $\mathbb{C}^{N \times N}$, and has full measure (i.e. its complement has measure 0).*

In other words, Proposition 3.4 says that *(almost) all SSMs are equivalent to a diagonal SSM*.

²Note that although we actually require the discretized matrices $(\bar{\mathbf{A}}, \bar{\mathbf{B}})$ when computing an SSM, conjugation commutes with discretization so we refer to \mathbf{A} in this section.

Moreover, diagonal SSMs are also highly structured in a way that addresses Problem 1 and Problem 2. In particular, computing $\bar{\mathbf{K}}$ (equation (2.7)) becomes a well-studied structured matrix multiplication with efficient time and space complexity.

3.2.1 S4D: Diagonal SSM Algorithms

Remark 3.6. *In the case of diagonal SSMs, \mathbf{A} is diagonal and we will overload notation so that \mathbf{A}_n denotes the entries of its diagonal. Recall that we assume the SISO case where $\mathbf{B} \in \mathbb{R}^{N \times 1}$ and $\mathbf{C} \in \mathbb{R}^{1 \times N}$, so we will also let $\mathbf{B}_n, \mathbf{C}_n$ directly index their elements.*

We now present S4D and address Problem 1 and Problem 2 for diagonal SSMs.

S4D Recurrence

Computing any discretization on diagonal SSMs is easy, because analytic functions on diagonal matrices reduce to performing them elementwise on their diagonal entries.

Performing MVM by a diagonal matrix is also easy, because it reduces to elementwise multiplications. Thus diagonal SSMs trivially satisfy Definition 3.1.

S4D Convolution Kernel: Vandermonde Matrix Multiplication

When \mathbf{A} is diagonal, computing the convolution kernel (2.8) becomes extremely simple:

$$\bar{\mathbf{K}}_\ell = \sum_{n=0}^{N-1} \mathbf{C}_n \bar{\mathbf{A}}_n^\ell \bar{\mathbf{B}}_n \implies \bar{\mathbf{K}} = (\bar{\mathbf{B}}^\top \circ \mathbf{C}) \cdot \mathcal{V}_L(\bar{\mathbf{A}}) \quad \text{where} \quad \mathcal{V}_L(\bar{\mathbf{A}})_{n,\ell} = \bar{\mathbf{A}}_n^\ell \quad (3.2)$$

where \circ is Hadamard product, \cdot is matrix multiplication, and \mathcal{V} is known as a **Vandermonde matrix**.

Unpacking this a little more, we can write $\bar{\mathbf{K}}$ as the following Vandermonde matrix-vector multiplication.

$$\bar{\mathbf{K}} = [\bar{\mathbf{B}}_0 \mathbf{C}_0 \dots \bar{\mathbf{B}}_{N-1} \mathbf{C}_{N-1}] \begin{bmatrix} 1 & \bar{\mathbf{A}}_0 & \bar{\mathbf{A}}_0^2 & \dots & \bar{\mathbf{A}}_0^{L-1} \\ 1 & \bar{\mathbf{A}}_1 & \bar{\mathbf{A}}_1^2 & \dots & \bar{\mathbf{A}}_1^{L-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \bar{\mathbf{A}}_{N-1} & \bar{\mathbf{A}}_{N-1}^2 & \dots & \bar{\mathbf{A}}_{N-1}^{L-1} \end{bmatrix}$$

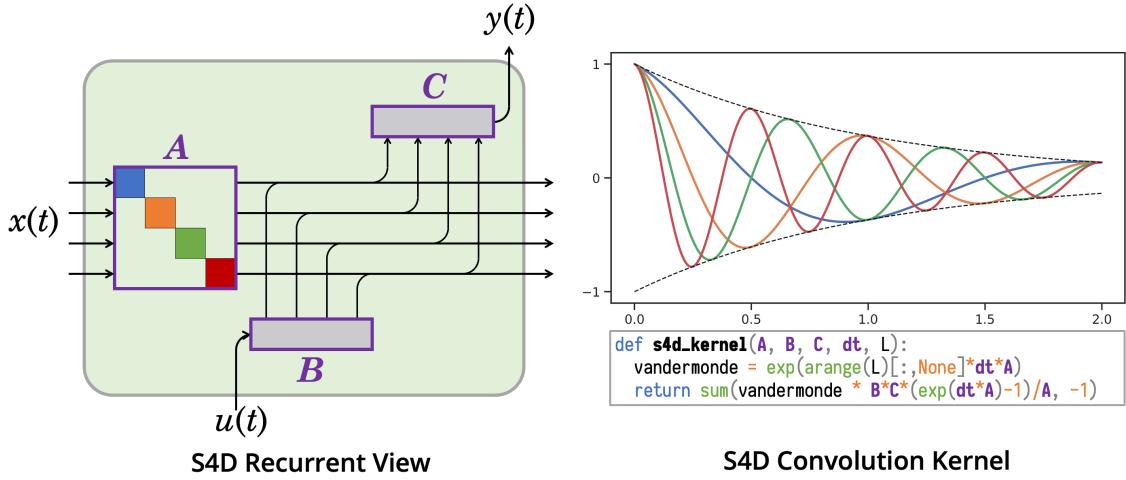


Figure 3.1: Diagonally structured SSMs (S4D) have very simple interpretations. (*Left*) The diagonal structure allows it to be viewed as a collection of 1-dimensional SSMs, or scalar recurrences. (*Right*) As a convolutional model, S4D has a simple interpretable convolution kernel which can be implemented in two lines of code. Colors denote independent 1-D SSMs; purple denotes trainable parameters.

Time and Space Complexity

The naive way to compute (3.2) is by materializing the Vandermonde matrix $\mathcal{V}_L(\bar{A})$ and performing a matrix multiplication, which requires $O(NL)$ time and space.

However, Vandermonde matrices are well-studied and theoretically the multiplication can be computed in $\tilde{O}(N + L)$ operations and $O(N + L)$ space [152].

3.2.2 Complete Implementation Example

The entire S4D method is very straightforward to implement, requiring just a few lines of code each for the parameterization and initialization, kernel computation, and full forward pass (Listing 1).

Finally, note that different combinations of parameterization choices can lead to slightly different implementations of the kernel. Figure 3.1 illustrates the S4D kernel with ZOH discretization which can be simplified even further to just *2 lines of code*.

```

def parameters(N, dt_min=1e-3, dt_max=1e-1):
    # Initialization
    # Geometrically uniform timescale [Chapter 5]
    log_dt = np.random.rand() * (np.log(dt_max)-np.log(dt_min)) + np.log(dt_min)
    # S4D-Lin initialization for (A, B) [Chapter 6]
    A = -0.5 + 1j * np.pi * np.arange(N//2)
    B = np.ones(N//2) + 0j
    # Variance preserving initialization [Chapter 5]
    C = np.random.randn(N//2) + 1j * np.random.randn(N)
    return log_dt, np.log(-A.real), A.imag, B, C

def kernel(L, log_dt, log_A_real, A_imag, B, C):
    # Discretization (e.g. bilinear transform)
    dt, A = np.exp(log_dt), -np.exp(log_A_real) + 1j * A_imag
    dA, dB = (1+dt*A/2) / (1-dt*A/2), dt*B / (1-dt*A/2)

    # Computation (Vandermonde matrix multiplication - can be optimized)
    # Return twice the real part - same as adding conjugate pairs
    return 2 * ((B*C) @ (dA[:, None] ** np.arange(L))).real

def forward(u, parameters):
    L = u.shape[-1]
    K = kernel(L, *parameters)
    # Convolve y = u * K using FFT
    K_f, u_f = np.fft.fft(K, n=2*L), np.fft.fft(u, n=2*L)
    return np.fft.ifft(K_f*u_f, n=2*L)[..., :L]

```

Listing 1: Full Numpy example of the parameterization and computation of a 1-channel S4D model.

3.3 The Diagonal Plus Low-Rank (DPLR) Parameterization

When possible, diagonal SSMs are ideal to use in practice because of their simplicity and flexibility. However, their strong structure can sometimes be too restrictive. In particular, Chapter 6 will show that the important class of SSMs based on HIPPO matrices (Chapters 4 and 5) cannot be numerically represented as diagonal SSMs, and instead motivates an extension of diagonal structure. While we defer this motivation to Part II, this section is a self-contained presentation of this structure from a computational perspective; beyond the connections to special SSMs in Part II, the ideas behind this parameterization and algorithms are of standalone interest theoretically, and have been used in later sequence models (Section 3.6.2).

This section defines an extension of diagonal SSMs to **diagonal plus low-rank (DPLR)** SSMs that is still computationally efficient. Our main technical results focus on developing this parameterization and showing how to efficiently compute all representations of the SSM (Section 2.3), in particular finding algorithms for Problem 1 and Problem 2.

Algorithm 1 S4 CONVOLUTION KERNEL (SKETCH)

Input: S4 parameters $\Lambda, P, Q, B, C \in \mathbb{C}^N$ and step size Δ **Output:** SSM convolution kernel $\bar{K} = \mathcal{K}_L(\bar{A}, \bar{B}, C)$ for $A = \Lambda - PQ^*$ (equation (2.8))

- 1: $\tilde{C} \leftarrow (\mathbf{I} - \bar{A}^L)^* C$ ▷ Truncate SSM generating function (SSMGF) to length L
 - 2: $\begin{bmatrix} k_{00}(\omega) & k_{01}(\omega) \\ k_{10}(\omega) & k_{11}(\omega) \end{bmatrix} \leftarrow [\tilde{C} \ Q]^* \left(\frac{2}{\Delta} \frac{1-\omega}{1+\omega} - \Lambda \right)^{-1} [B \ P]$ ▷ Black-box Cauchy kernel
 - 3: $\hat{K}(\omega) \leftarrow \frac{2}{1+\omega} [k_{00}(\omega) - k_{01}(\omega)(1 + k_{11}(\omega))^{-1} k_{10}(\omega)]$ ▷ Woodbury Identity
 - 4: $\hat{K} = \{\hat{K}(\omega) : \omega = \exp(2\pi i \frac{k}{L})\}$ ▷ Evaluate SSMGF at all roots of unity $\omega \in \Omega_L$
 - 5: $\bar{K} \leftarrow \text{iFFT}(\hat{K})$ ▷ Inverse Fourier Transform
-

Section 3.3.1 gives an overview of the key technical components of our approach and formally defines the S4-DPLR parameterization. Section 3.3.2 sketches the main results, showing that S4 is asymptotically efficient (up to log factors) for sequence models. Proofs are in Appendix C.1.

3.3.1 Overview of the DPLR State Space Kernel Algorithm

Despite the seemingly small extension from diagonal to DPLR matrices, the addition of this low-rank term makes computing with the matrix much more difficult. In particular, unlike for a diagonal matrix, computing powers of a DPLR matrix for equation (2.8) is still slow (as for unstructured matrices) and not easily optimized. We overcome this bottleneck by simultaneously applying three new techniques.

- Instead of computing \bar{K} directly, we compute its spectrum by evaluating its **truncated generating function** $\sum_{j=0}^{L-1} \bar{K}_j \zeta^j$ at the roots of unity ζ . \bar{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** (Proposition A.2) which reduces $(A + PQ^*)^{-1}$ in terms of 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 [154, 153].

3.3.2 S4-DPLR Algorithms and Computational Complexity

Our algorithms are optimal for both recurrent and convolutional representations, satisfying Definitions 3.1 and 3.2.

Theorem 3.5 (S4 Recurrence). *Given any step size Δ , computing one step of the recurrence (2.5) can be done in $O(N)$ operations where N is the state size.*

Theorem 3.5 follows from the fact that the inverse of a DPLR matrix is also DPLR (e.g. which also follows from the Woodbury identity). This implies that the discretized matrix $\bar{\mathbf{A}}$ is the product of two DPLR matrices and thus has $O(N)$ matrix-vector multiplication. Appendix C.1.1 computes $\bar{\mathbf{A}}$ in closed DPLR form.

Theorem 3.6 (S4 Convolution). *Given any step size Δ , computing the SSM convolution filter $\bar{\mathbf{K}}$ can be reduced to 4 Cauchy multiplies, requiring only $\tilde{O}(N + L)$ operations and $O(N + L)$ space.*

Appendix C.1, Definition C.5 formally defines Cauchy matrices, which are related to rational interpolation problems. Computing with Cauchy matrices is an extremely well-studied problem in numerical analysis, with both fast arithmetic and numerical algorithms based on the famous Fast Multipole Method (FMM) [152, 154, 153]. The computational complexities of these algorithms under various settings are described in Appendix C.1, Proposition C.6.

We emphasize that Theorem 3.6 is the technical core of S4, and its algorithm is the very motivation of the DPLR parameterization. This algorithm is sketched in Algorithm 1.

3.3.3 Hurwitz (Stable) DPLR Form

Independent of the algorithmic details of computing S4-DPLR, we will use one modification to the basic DPLR parameterization to ensure stability of the state space model. In particular, **Hurwitz matrices** (also called stable matrices) are a class of matrices ensuring that the SSM (2.1) is asymptotically stable.

Definition 3.7. A Hurwitz matrix \mathbf{A} is one where every eigenvalue has negative real part.

From the perspective of discrete-time SSMs, we can easily see why \mathbf{A} needs to be a Hurwitz matrix from first principles with the following simple observations. First, unrolling the RNN mode (2.5) involves powering up $\bar{\mathbf{A}}$ repeatedly, which is stable if and only if all eigenvalues of $\bar{\mathbf{A}}$ lie inside or on the (complex) unit disk. Second, the transformation (2.4) (for either

Bilinear or ZOH discretizations) maps the complex left half plane to the complex unit disk. Therefore computing the RNN mode of an SSM (e.g. for autoregressive inference) requires \mathbf{A} to be a Hurwitz matrix.

From the continuous perspective, another one way to see this is by the fact that linear ODEs (2.1a) solves to an exponential. Concretely, we can also see that the equivalent convolution form (2.2) has impulse response $K(t) = \mathbf{C}e^{t\mathbf{A}}\mathbf{B}$ which blows up to ∞ as $t \rightarrow \infty$.

However, controlling the spectrum of a general DPLR matrix is difficult; in early versions of S4, we found that unrestricted DPLR matrices generally became non-Hurwitz after training (and thus could not be used in unbounded recurrent mode) (c.f. Section 8.4.2).

To address this, we use a minor modification of DPLR matrices which we call the **Hurwitz DPLR form**, which uses the parameterization $\Lambda - \mathbf{P}\mathbf{P}^*$ instead of $\Lambda + \mathbf{P}\mathbf{Q}^*$. This amounts to essentially tying the parameters $\mathbf{Q} = -\mathbf{P}$. Note that this is still technically DPLR, so we can use the S4-DPLR algorithms as a black box.

Next, we discuss how this parameterization makes S4 stable. The high-level idea is that stability of SSMs involves the spectrum of the state matrix \mathbf{A} , which is more easily controlled because $-\mathbf{P}\mathbf{P}^*$ is a negative semidefinite matrix (i.e. we know the signs of its spectrum).

Lemma 3.8. *A matrix $\mathbf{A} = \Lambda - \mathbf{P}\mathbf{P}^*$ is Hurwitz if all entries of Λ have negative real part.*

Proof. We first observe that if $\mathbf{A} + \mathbf{A}^*$ is negative semidefinite (NSD), then \mathbf{A} is Hurwitz. This follows because $0 > v^*(\mathbf{A} + \mathbf{A}^*)v = (v^*\mathbf{A}v) + (v^*\mathbf{A}v)^* = 2\Re(v^*\mathbf{A}v) = 2\lambda$ for any (unit length) eigenpair (λ, v) of \mathbf{A} .

Next, note that the condition implies that $\Lambda + \Lambda^*$ is NSD (it is a real diagonal matrix with non-positive entries). Since the matrix $-\mathbf{P}\mathbf{P}^*$ is also NSD, then so is $\mathbf{A} + \mathbf{A}^*$. \square

Lemma 3.8 implies that with the Hurwitz DPLR representation, controlling the spectrum of the learned \mathbf{A} matrix becomes simply controlling the the diagonal portion Λ . This is a far easier problem than controlling a general DPLR matrix, and can be enforced by regularization or reparameterization (Section 3.4.2).

We make two final remarks about this representation. These empirical considerations are mentioned and ablated in Section 8.4.2.

Remark 3.7. *The Hurwitz DPLR form $\mathbf{\Lambda} - \mathbf{P}\mathbf{P}^*$ has fewer parameters and is technically less expressive than the unrestricted DPLR form $\mathbf{\Lambda} + \mathbf{P}\mathbf{Q}^*$, but it does not impact model performance empirically [67].*

Remark 3.8. *Potential stability issues only arise when using S4 in certain contexts such as autoregressive generation, because S4’s convolutional mode during training does not involve powering up $\bar{\mathbf{A}}$ and thus does not strictly require a Hurwitz matrix. In practice we still always use the Hurwitz DPLR form out of principle.*

3.4 Additional Parameterization Details

Aside from the most important parameterization choice of the structure on \mathbf{A} , there are several other details and options in defining S4 models.

3.4.1 Discretization

Section 2.3.1 (and Appendix A.1) discusses several possible discretization methods, which offer a tradeoff between simplicity and accuracy. For example, the Euler method reduces Problem 1 to simply requiring structure on \mathbf{A} , whereas accurate rules such as ZOH and bilinear require additional structure (see Section 3.1.1). The DPLR parameterization and algorithms (Section 3.3) are specialized to the generalized bilinear transforms (Appendix A.1) and do not work for ZOH. This is because the inverse of a DPLR matrix is also DPLR, but there is no obvious structure for the matrix exponential of a DPLR matrix.

On the other hand, diagonal matrices are particularly simple because any matrix function reduces to elementwise scalar functions on the diagonal. S4D therefore disentangles the discretization method from the kernel computation (equation (3.2)), so that any discretization method is feasible.

Ultimately, however, although there may be theoretical reasons to prefer one discretization over another, there is little empirical difference between ZOH and bilinear and unlikely to be room for improvement with alternative methods. Prior and related works have used both methods without meaningful difference; we also include a small ablation in Section 7.5.3.

3.4.2 Parameterization of \mathbf{A}

By the discussion in Section 3.3.3, \mathbf{A} should only have eigenvalues with non-positive real part. This is a serious constraint that affects the stability of the model, especially when using the SSM as an autoregressive generative model (Chapter 8). For diagonal SSMs, we can easily constrain the real part of \mathbf{A} to be negative (also known as the left-half plane condition in classical controls), by parameterizing the real part inside an exponential function $\mathbf{A} = -\exp(\mathbf{A}_{Re}) + i \cdot \mathbf{A}_{Im}$.

We note that instead of \exp , any activation function can be used as long as its range is bounded on one side, such as ReLU, softplus, etc. This does not make a significant empirical difference, and a small ablation is included in Section 7.5.3.

For DPLR parameterization, on top of constraining the diagonal part of \mathbf{A} (denoted $\mathbf{\Lambda}$ in this chapter), \mathbf{A} should also be represented in the Hurwitz DPLR form (Section 3.3.3). Some empirical analysis of what happens when \mathbf{A} is not constrained to be Hurwitz (stable) is shown in Chapter 8.

3.4.3 Parameterization of \mathbf{B} and \mathbf{C}

Another choice in the parameterization is how to represent \mathbf{B} and \mathbf{C} . For S4D, note that the computation of the final discrete convolution kernel $\bar{\mathbf{K}}$ depends only on the elementwise product $\mathbf{B} \circ \mathbf{C}$ (equation (3.2)). Therefore this product could be parameterized directly, instead of learning two independent parameters. However, we observe that this is equivalent to keeping separate \mathbf{B} and \mathbf{C} , and simply freezing $\mathbf{B} = \mathbf{1}$ while training \mathbf{C} . Thus S4D has a choice of whether to combine \mathbf{B} and \mathbf{C} , but equivalently it is a choice whether or not to train \mathbf{B} in general.

3.4.4 Complex Numbers

Proposition 3.4 says that *(almost) all real SSMs are equivalent to a complex diagonal SSM*, while clearly diagonal real matrices are not as expressive as dense real matrices. Therefore it is critical to use complex-valued matrices in order to use diagonal SSMs. Similarly, DPLR SSMs should be parameterized over \mathbb{C} instead of \mathbb{R} .

3.4.5 Conjugate Symmetry

Finally, we make note of a minor parameterization detail when moving from real to complex numbers. Note that we ultimately care about sequence transformations over *real* numbers.

In this case, note that the state x_k is a vector \mathbb{R}^N , and similarly \mathbf{B} and \mathbf{C} would consist of N real parameters.

However, if using complex numbers, this effectively doubles the state dimension and the number of parameters in \mathbf{B}, \mathbf{C} . Furthermore, when using a complex SSM, the output of the SSM is not guaranteed to be real even if the input is real, and similarly the convolution kernel (3.2) will in general be complex.

To resolve this discrepancy, note that when diagonalizing a real SSM into a complex SSM (see Proposition 3.4), the resulting parameters always occur in **conjugate pairs**. Therefore we can throw out half of the parameters.

In other words, to parameterize a real SSM of state size N , we can instead parameterize a complex SSM of state size $\frac{N}{2}$, and implicitly add back the conjugate pairs of the parameters. This ensures that the total state size and parameter count is actually the equivalent of N real numbers, and also guarantees that the output of the kernel is real. The implementation of this is very simple; the sum in (3.2) will implicitly include the conjugate pairs of $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and therefore resolve to twice the real part of the original sum.

An example implementation of this detail can be seen in Listing 1.

3.5 Comparing S4-Diag and S4-DPLR

3.5.1 Expressivity

Although complex diagonal SSMs theoretically represent (almost) all real SSMs, we emphasize that Proposition 3.4 is about *expressivity* which does not guarantee strong performance of a trained model after optimization. For example, Section 7.5 has ablations showing that parameterizing \mathbf{A} as a dense real matrix or diagonal complex matrix, which are both fully expressive classes, still perform poorly if randomly initialized.

Second, Proposition 3.4 does not take into account numerical representations of data. This is in fact the primary motivation for the DPLR representation: Chapter 6 shows that a particular important SSM cannot be diagonalized stably. In Section 7.5.4, we also show that two different initializations with the *same spectrum* (i.e., can be transformed to the same diagonal \mathbf{A}) can have very different empirical performance.

3.5.2 Initialization of Parameters

The above observations point to the need for properly instantiating the SSM parameters, which is achieved by special initializations. These instantiations are the subject of Part II, while this chapter has focused solely on the parameterization and computation of S4 variants.

The initialization is not strictly part of the computational aspects and so the definition of S4 is self-contained in this chapter; we point to Section 7.1 which has succinct descriptions and references for the rest of these modeling details.

3.5.3 Computational Complexities

The computational core of S4D’s convolution kernel is the Vandermonde matrix multiplication (Equation (3.2)), while the core of S4’s kernel is the Cauchy matrix multiplication (Sections 3.3.1 and 3.3.2 and Appendix C.1.2). In fact, Vandermonde matrices are closely related to Cauchy matrices, and they have identical computation complexity [152].

Lemma 3.9. *The time and space complexity of computing the kernel of diagonal SSMs is equal to that of computing DPLR SSMs.*

However, although these structured kernels have theoretically efficient algorithms, these algorithms are currently not implemented for modern hardware accelerators due to a lack of previous applications that require them. A simpler implementation is to compute the matrix multiplication (e.g. (3.2)) with naive summation (using $O(NL)$ operations), but without materializing the full Vandermonde or Cauchy matrix (using linear $O(N + L)$ space).³ This is quite fast and hardware-efficient on accelerators such as GPUs, and avoids the space issues of general SSMs – the main computational issue discussed in Section 3.1.2.

3.5.4 Overall Parameterization

Table 3.1: **(Parameterization of Structured SSMs.)** Both diagonal (S4D) and DPLR (S4) structures are efficient, but S4 is more general while S4D is simpler and more flexible.

Model	Structure	Core Kernel	Discretization	Combine (\mathbf{B}, \mathbf{C})	Init. of (\mathbf{A}, \mathbf{B})
S4D	Diagonal	Vandermonde	any	optional	various (e.g. LegS, Lin)
S4	DPLR	Cauchy	Bilinear	no	various (e.g. LegS, FouT)

³This may require implementing a custom kernel in some modern deep learning frameworks such as PyTorch to achieve the space savings.

Table 3.2: (**Sequence model complexities.**) Complexity of various sequence models in terms of sequence length (L), batch size (B), and hidden dimension (H); tildes denote log factors. Metrics are parameter count, training computation, training space requirement, training parallelizability, and inference computation (for 1 sample and time-step). For simplicity, the state size N of S4 is tied to H . Bold denotes model is theoretically best for that metric among these comparisons. Convolutions are efficient for training while recurrence is efficient for inference, while SSMs combine the strengths of both. (Appendix C.2 provides a more detailed version of this table, where N and H are decoupled, and other SSM variants are included.)

	Convolution ⁴	Recurrence	Attention	S4
Parameters	LH	\mathbf{H}^2	\mathbf{H}^2	\mathbf{H}^2
Training	$\tilde{L}\mathbf{H}(B + H)$	BLH^2	$B(L^2H + LH^2)$	$\mathbf{BH}(\tilde{H} + \tilde{L}) + B\tilde{L}H$
Space	BLH	BLH	$B(L^2 + HL)$	BLH
Parallel	Yes	No	Yes	Yes
Inference	LH^2	\mathbf{H}^2	$L^2H + H^2L$	\mathbf{H}^2

Table 3.1 compares S4 and S4D, which each have a core structure and kernel computation, but various other parameterization choices. Unless otherwise specified (e.g. for an empirical ablation), we define S4D to match the parameterization of S4 with bilinear discretization and separate (trainable) (\mathbf{B}, \mathbf{C}) parameters. Note that S4D is then exactly the same model as S4 with low-rank component set to 0, i.e. they define the same function $u \mapsto y$ and the algorithms in Section 3.2 and Section 3.3 would compute the same answer (up to numerical imprecision in floating point computations).

In our empirical study of S4 in Section 7.5, we ablate many of these parameterization choices, including the overall S4 vs. S4D models (i.e. DPLR vs diagonal representations).

3.6 Discussion

3.6.1 Comparison between S4 and Other Sequence Models

As defined in this chapter, S4 is a SISO (single-input single-output) SSM that maps 1-D signals. We refer to Section 2.3.5 for its definition on inputs with multiple channels. Table 3.2 compares the computational complexity of S4 against the other standard deep sequence models.

Comparisons of computational speed in practice are included in the experiments in Section 7.4.3. For example, on larger model sizes, S4-DPLR is up to $30\times$ faster with $400\times$ less

⁴Refers to global (in the sequence length) and depthwise-separable convolutions, similar to the convolution version of S4.

memory usage than an unstructured SSM.

3.6.2 Other Structured SSMs

The original version of S4 is associated with DPLR (or diagonal) SSMs. However, the concept of structured SS(S)Ms that it introduced as a way of overcoming computational bottlenecks of general SSMs (Section 3.1) is more fundamental. There are many families of structured matrices [152, 41], and other flavors of structured SSMs have appeared based on S4. We mention a few of these.

Diagonal and DPLR These are the original variants of S4 presented in this chapter and thesis.

Shift Suppose A is fixed to be the **shift matrix**

$$\bar{A} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

This was proposed as a component in the H3 architecture [37], along with an algorithm to compute the convolution kernel efficiently.

Note however that this shift-SSM is *essentially identical to a vanilla local convolution*. To see this, consider the SISO case and setting $B = e_0$ to be the basis vector. Notice that $AB = e_1, A^2B = e_2, \dots, A^kB = e_k, \dots, A^N B = \mathbf{0}$. Therefore the kernel \bar{K} is just equal to C (padded with 0's).

In other words, the SISO shift-SSM with $B = e_0$ is exactly identical to a standard local convolution, in particular a **depthwise causal convolution** (see also . The more general case is similar, and the efficient kernel algorithm for shift SSM reduces to just a few convolutions.

In other words: *standard local convolutions of width k are precisely shift SSMs with state size k*. Writing it in this way also immediately exposes the fact that local convolutions have fast recurrent inference, which is the subject of dedicated papers (e.g. an efficient cached implementation of WaveNet in Paine et al. [151]. This illustrates the power of the SSM abstraction which interfaces between convolutions and recurrence!

Companion A generalization of shift matrices are **companion matrices**

$$\bar{\mathbf{A}} = \begin{bmatrix} 0 & 0 & 0 & -c_0 \\ 1 & 0 & 0 & -c_1 \\ 0 & 1 & 0 & -c_2 \\ 0 & 0 & 1 & -c_3 \end{bmatrix}$$

These appear in many fields of algebra as a canonical form representing the characteristic polynomial $c_3x^3 + c_2x^2 + c_1x + c_0$, and as an SSM were applied to time series applications in Zhang et al. [239]. Just as DPLR is a generalization of diagonal matrices by adding a low-rank term, note that a companion matrix is a rank-1 addition to the shift matrix. By using a technique based on the DPLR algorithm from Section 3.3, an “S4-Comp” kernel can be reduced to the “S4-Shift” case to compute its convolution kernel efficiently.

Part II

Orthogonal State Space Models for Online Memorization

Chapter 4

HIPPO: Continuous Memory with Optimal Polynomial Projections

Part I primarily focused on *computational* aspects of state space models, such as their different representations and making them efficient through imposing additional structure. Part II now tackles the main remaining challenge about how to *model* complex data, answering questions such as:

- How can SSMs be instantiated to be able to model long-range dependencies?
- What does the latent state $x(t)$ actually represent, and can the SSM be defined so that it has a useful and interpretable meaning?

This chapter takes a first-principles approach toward these questions, completely independent of the state space model terminology. It will be connected back to SSMs in the next two chapters.

Remark 4.1. *To emphasize the independence from SSMs, the material in this chapter uses different notation: instead of $u(t)$ to denote inputs and $x(t)$ for the state, it uses $f(t)$ to denote an input function and $c(t)$ to denote a state of “polynomial coefficients”. This is notation from the originally published HIPPO paper.*

4.1 Introduction

Modeling and learning from sequential data is a fundamental problem in modern machine learning, underlying tasks such as language modeling, speech recognition, video processing, and reinforcement learning. A core aspect of modeling long-term and complex temporal dependencies is *memory*, or storing and incorporating information from previous time steps. The challenge is learning a representation of the entire cumulative history using bounded storage, which must be updated online as more data is received.

One established approach is to model a state that evolves over time as it incorporates more information. The deep learning instantiation of this approach is the recurrent neural network (RNN), which is known to suffer from a limited memory horizon [86, 95, 156] (e.g., the “vanishing gradients” problem). Although various heuristics have been proposed to overcome this, such as gates in the successful LSTM and GRU [86, 29], or higher-order frequencies in the recent Fourier Recurrent Unit [238] and Legendre Memory Unit (LMU) [218], a unified understanding of memory remains a challenge. Furthermore, existing methods generally require priors on the sequence length or timescale and are ineffective outside this range [198, 218]; this can be problematic in settings with distribution shift (e.g. arising from different instrument sampling rates in medical data [180, 185]). Finally, many of them lack theoretical guarantees on how well they capture long-term dependencies, such as gradient bounds. To design a better memory representation, we would ideally (i) have a unified view of these existing methods, (ii) be able to address dependencies of any length without priors on the timescale, and (iii) have a rigorous theoretical understanding of their memory mechanism.

Our insight is to phrase *memory* as a technical problem of **online function approximation** where a function $f(t) : \mathbb{R}_+ \rightarrow \mathbb{R}$ is summarized by storing its optimal coefficients in terms of some basis functions. This approximation is evaluated with respect to a measure that specifies the importance of each time in the past. Given this function approximation formulation, orthogonal polynomials (OPs) emerge as a natural basis since their optimal coefficients can be expressed in closed form [26]. With their rich and well-studied history [197], along with their widespread use in approximation theory [212] and signal processing [166], OPs bring a library of techniques to this memory representation problem. We formalize a framework, **HIPPO** (high-order polynomial projection operators), which produces operators that project arbitrary functions onto the space of orthogonal polynomials with respect to a given measure. This general framework allows us to analyze several families of

measures, where this operator, as a closed-form ODE or linear recurrence, allows fast incremental updating of the optimal polynomial approximation as the input function is revealed through time.

By posing a formal optimization problem underlying recurrent sequence models, the HIPPO framework (Section 4.2) generalizes and explains previous methods, unlocks new methods appropriate for sequential data at different timescales, and comes with several theoretical guarantees.

- (i) For example, with a short derivation we exactly recover as a special case the LMU [218] (Section 4.2.3), which proposes an update rule that projects onto fixed-length sliding windows through time. HIPPO also sheds new light on classic techniques such as the gating mechanism of LSTMs and GRUs, which arise in one extreme using only low-order degrees in the approximation (Section 4.2.5).
- (ii) By choosing more suitable measures, HIPPO yields a novel mechanism (Scaled Legendre, or LegS) that always takes into account the function’s full history instead of a sliding window. This flexibility removes the need for hyperparameters or priors on the sequence length, allowing LegS to generalize to different input timescales.
- (iii) The connections to dynamical systems and approximation theory allows us to show several theoretical benefits of HIPPO-LegS: invariance to input timescale, asymptotically more efficient updates, and bounds on gradient flow and approximation error (Section 4.3).

In Section 4.4, we validate HIPPO’s theory, including computational efficiency and scalability, allowing fast and accurate online function reconstruction over millions of time steps (Section 4.4.1). We also demonstrate the timescale robustness of HIPPO-LegS on a novel trajectory classification task, where it is able to generalize to unseen timescales and handle missing data whereas RNN and neural ODE baselines fail (Section 4.4.3).

4.2 High-order Polynomial Projection Operators

We motivate the problem of online function approximation with projections as an approach to learning memory representations (Section 4.2.1). Section 4.2.2 describes the general HIPPO framework to derive memory updates, including a precise definition of the technical problem we introduce, and an overview of our approach to solving it. Section 4.2.3 instantiates the framework to recover the LMU demonstrating the generality of the HIPPO

framework. Section 4.2.4 discusses how to convert the main continuous-time results into practical discrete versions. Finally in Section 4.2.5 we show how gating in RNNs is an instance of HIPPO memory.

4.2.1 HIPPO Problem Setup

Given an input function $f(t) \in \mathbb{R}$ on $t \geq 0$, many problems require operating on the cumulative *history* $f_{\leq t} := f(x) |_{x \leq t}$ 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. Thus, our goal is to maintain (online) this compressed representation of the history. In order to specify this problem fully, we require two ingredients: a way to quantify the approximation, and a suitable subspace.

Function Approximation with respect to a Measure. Assessing the quality of an approximation requires defining a distance in function space. Any probability measure μ on $[0, \infty)$ equips the space of square integrable functions with inner product

$$\langle f, g \rangle_\mu = \int_0^\infty f(x)g(x) d\mu(x),$$

inducing a Hilbert space structure \mathcal{H}_μ and corresponding norm $\|f\|_{L_2(\mu)} = \langle f, f \rangle_\mu^{1/2}$.

Polynomial Basis Expansion. Any N -dimensional subspace \mathcal{G} of this function space is a suitable candidate for the approximation. The parameter N corresponds to the order of the approximation, or the size of the compression; the projected history can be represented by the N coefficients of its expansion in any basis of \mathcal{G} . For the remainder of this paper, we use the polynomials as a natural basis, so that \mathcal{G} is the set of polynomials of degree less than N . We note that the polynomial basis is very general; for example, the Fourier basis $\sin(nx), \cos(nx)$ can be seen as polynomials on the unit circle ($e^{2\pi i x}$)ⁿ. In Appendix D.1 and later in Chapter 5, we additionally formalize a more general framework that allows different bases other than polynomials by tilting the measure with another function.

Online Approximation. Since we care about approximating $f_{\leq t}$ for every time t , we also let the measure vary through time. For every t , let $\mu^{(t)}$ be a measure supported on $(-\infty, t]$ (since $f_{\leq t}$ is only defined up to time t). Overall, we seek some $g^{(t)} \in \mathcal{G}$ that minimizes

$\|f_{\leq t} - g^{(t)}\|_{L_2(\mu^{(t)})}$. Intuitively, the measure μ controls the importance of various parts of the input domain, and the basis defines the allowable approximations. The challenge is how to solve the optimization problem in closed form given $\mu^{(t)}$, and how these coefficients can be maintained online as $t \rightarrow \infty$.

4.2.2 General HIPPO framework

We provide a brief overview of the main ideas behind solving this problem, which provides a surprisingly simple and general strategy for many measure families $\mu^{(t)}$. This framework builds upon a rich history of the well-studied **orthogonal polynomials** and related transforms in the signal processing literature. Our formal abstraction (Definition 4.1) departs from prior work on sliding transforms in several ways, which we discuss in detail in the Related Work chapter. For example, our concept of the time-varying measure allows choosing $\mu^{(t)}$ more appropriately, which will lead to solutions with qualitatively different behavior. Appendix D.1 contains the full details and formalisms of our framework.

Calculating the projection through continuous dynamics. As mentioned, the approximated function can be represented by the N coefficients of its expansion in any basis; the first key step is to choose a suitable basis $\{g_n\}_{n < N}$ of \mathcal{G} . Leveraging classic techniques from approximation theory, a natural basis is the set of orthogonal polynomials for the measure $\mu^{(t)}$, which forms an orthogonal basis of the subspace. Then the coefficients of the optimal basis expansion are simply $c_n^{(t)} := \langle f_{\leq t}, g_n \rangle_{\mu^{(t)}}$.

The second key idea is to differentiate this projection in t , where differentiating through the integral (from the inner product $\langle f_{\leq t}, g_n \rangle_{\mu^{(t)}}$) will often lead to a self-similar relation allowing $\frac{d}{dt}c_n(t)$ to be expressed in terms of $(c_k(t))_{k \in [N]}$ and $f(t)$. Thus the coefficients $c(t) \in \mathbb{R}^N$ should evolve as an ODE, with dynamics determined by $f(t)$.

The HIPPO abstraction: online function approximation.

Definition 4.1. *Given a time-varying measure family $\mu^{(t)}$ supported on $(-\infty, t]$, an N -dimensional subspace \mathcal{G} of polynomials, and a continuous function $f: \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$, 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 , $f_{\leq t} := f(x) |_{x \leq t}$, and maps it to a polynomial $g^{(t)} \in \mathcal{G}$, that minimizes the approximation error $\|f_{\leq t} - g^{(t)}\|_{L_2(\mu^{(t)})}$.

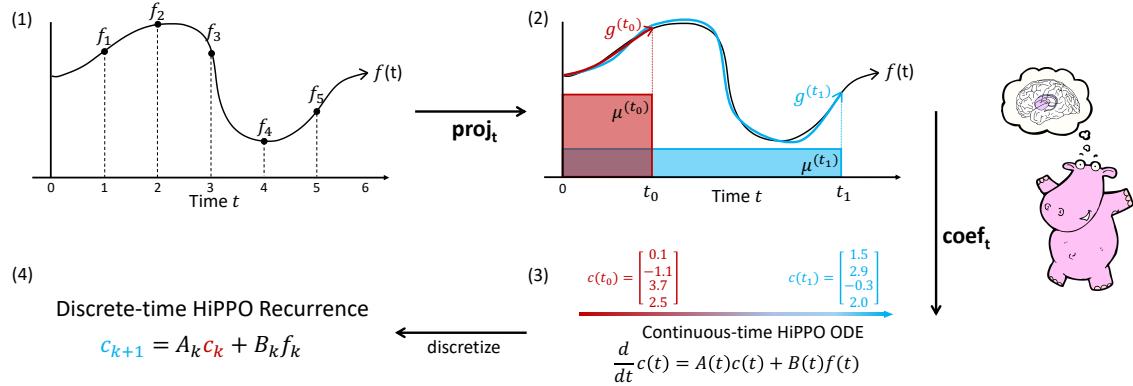


Figure 4.1: (**Illustration of the HIPPO framework.**) (1) For any function f , (2) at every time t there is an optimal projection $g^{(t)}$ of f onto the space of polynomials, with respect to a measure $\mu^{(t)}$ weighing the past. (3) For an appropriately chosen basis, the corresponding coefficients $c(t) \in \mathbb{R}^N$ representing a compression of the history of f satisfy linear dynamics. (4) Discretizing the dynamics yields an efficient closed-form recurrence for online compression of time series $(f_k)_{k \in \mathbb{N}}$.

(2) $\text{coef}_t : \mathcal{C} \rightarrow \mathbb{R}^N$ 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 $f : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$ to the optimal projection coefficients $c : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^N$, i.e. $(\text{hippo}(f))(t) = \text{coef}_t(\text{proj}_t(f))$.

For each t , the problem of optimal projection $\text{proj}_t(f)$ is well-defined by the above inner products, but this is intractable to compute naively. Our derivations (Appendix D.2) will show that the coefficient function $c(t) = \text{coef}_t(\text{proj}_t(f))$ has the form of an ODE satisfying $c'(t) = \mathbf{A}(t)c(t) + \mathbf{B}(t)f(t)$ for some $\mathbf{A}(t) \in \mathbb{R}^{N \times N}$, $\mathbf{B}(t) \in \mathbb{R}^{N \times 1}$. Thus our results show how to tractably obtain $c^{(t)}$ *online* by solving an ODE, or more concretely by running a discrete recurrence. When discretized, HIPPO takes in a sequence of real values and produces a sequence of N -dimensional vectors.

Figure 4.1 illustrates the overall framework when we use uniform measures. Next, we give our main results showing **hippo** for several concrete instantiations of the framework.

4.2.3 High Order Projection: Measure Families and HIPPO ODEs

Our main theoretical results are instantiations of HIPPO for various measure families $\mu^{(t)}$. The unified perspective on memory mechanisms allows us to derive these closed-form solutions with the same strategy (proofs in Appendix D.2). In this section, we focus on an example of a natural sliding window measure and the corresponding projection operator, which recreates the Legendre Memory Unit (LMU) [218] in a principled way and characterizes its limitations.

The **translated Legendre (LegT)** measures assign uniform weight to the most recent history $[t - \theta, t]$. There is a hyperparameter θ representing the length of the sliding window, or the length of history that is being summarized.

$$\text{LegT} : \mu^{(t)}(x) = \frac{1}{\theta} \mathbb{I}_{[t-\theta, t]}(x)$$

Theorem 4.2. *For LegT, the hippo operator satisfying Definition 4.1 is given by the linear time-invariant (LTI) ODE $c'(t) = -\mathbf{A}c(t) + \mathbf{B}f(t)$, where $\mathbf{A} \in \mathbb{R}^{N \times N}$ and $\mathbf{B} \in \mathbb{R}^{N \times 1}$:*

LegT:

$$\mathbf{A}_{nk} = \frac{1}{\theta} \cdot -(2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} \cdot \begin{cases} 1 & n \geq k \\ (-1)^{n-k} & n \leq k \end{cases} \quad \text{and} \quad \mathbf{B}_n = \frac{1}{\theta} (2n+1)^{\frac{1}{2}} \quad (4.1)$$

Equation (4.1) proves the LMU update [218, equation (1)] (up to scaling factors, discussed in the derivation Appendix D.2.1). Additionally, our derivation shows that outside of the projections, there is another source of approximation. This sliding window update rule requires access to $f(t - \theta)$, which is no longer available; it instead assumes that the current coefficients $c(t)$ are an accurate enough model of the function $f(x)_{x \leq t}$ that $f(t - \theta)$ can be recovered.

4.2.4 HIPPO Recurrences: from Continuous- to Discrete- Time

Since actual data is inherently discrete (e.g. sequences and time series), the continuous-time HIPPO operators can be discretized using the techniques in Section 2.3.1 and Appendix A.1 to become discrete-time linear recurrences.

In the continuous case, these operators consume an input function $f(t)$ and produce an output function $c(t)$. The discrete time case (i) consumes an input sequence $(f_k)_{k \in \mathbb{N}}$, (ii)

implicitly defines a function $f(t)$ where $f(k \cdot \Delta) = f_k$ for some step size Δ , (iii) produces a function $c(t)$ through the ODE dynamics, and (iv) discretizes back to an output sequence $c_k := c(k \cdot \Delta)$.

Finally, we note that this provides a way to seamlessly handle timestamped data, even with missing values: the difference between timestamps indicates the (adaptive) Δ to use in discretization [25].

4.2.5 Low Order Projection: Memory Mechanisms of Gated RNNs

As a special case, we consider what happens if we do not incorporate higher-order polynomials in the projection problem. Specifically, if $N = 1$, then the (Euler) discretized version of HIPPO-LegT becomes $c(t + \Delta) = c(t) + \Delta(-\mathbf{A}c(t) + \mathbf{B}f(t)) = (1 - \Delta)c(t) + \Delta f(t)$, since $\mathbf{A} = \mathbf{B} = 1$. If the inputs $f(t)$ can depend on the hidden state $c(t)$ and the discretization step size Δ is chosen adaptively (as a function of input $f(t)$ and state $c(t)$), as in RNNs, then this becomes exactly a *gated* RNN. For instance, by stacking multiple units in parallel and choosing a specific update function, we obtain the GRU update cell as a special case. In contrast to HIPPO which uses one hidden feature and projects it onto high order polynomials, these models use many hidden features but only project them with degree 1. This view sheds light on these classic techniques by showing how they can be derived from first principles.

Remark 4.2. See also Lemma 2.16, which provides a very similar discussion to this observation.

4.3 LegS: Scaled Measures for Timescale Robustness

Exposing the tight connection between online function approximation and memory allows us to produce memory mechanisms with better theoretical properties, simply by choosing the measure appropriately. Although sliding windows are common in signal processing, a more intuitive approach for memory should *scale* the window over time to avoid forgetting.

Our novel **scaled Legendre measure (LegS)** assigns uniform weight to all history $[0, t]$:

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

Simply by specifying the desired measure, specializing the HIPPO framework (Sections 4.2.2,

[4.2.4](#)) yields a new memory mechanism (proof in Appendix [D.2.2](#)).

Theorem 4.3. *The continuous- [\(4.2\)](#) and discrete- [\(4.3\)](#) time dynamics for HIPPO-LegS are:*

$$\frac{d}{dt} c(t) = -\frac{1}{t} \mathbf{A}c(t) + \frac{1}{t} \mathbf{B}f(t) \quad (4.2)$$

$$c_{k+1} = \left(1 - \frac{\mathbf{A}}{k}\right) c_k + \frac{1}{k} \mathbf{B}f_k \quad (4.3)$$

$$\mathbf{A}_{nk} = \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases}$$

$$\mathbf{B}_n = (2n+1)^{\frac{1}{2}}$$

We show that HIPPO-LegS enjoys favorable theoretical properties: it is invariant to input timescale, is fast to compute, and has bounded gradients and approximation error. All proofs are in Appendix [D.3](#).

Note that unlike the majority of results in this thesis, [\(4.2\)](#) is not time-invariant (LTI) because of the factor of $1/t$. In fact, this type of dynamical system also has many nice properties; some of our theoretical results such as Lemma [4.5](#) and Lemma [4.7](#) are true more broadly of these systems. We are unaware of a standard name for this type of ODE.

Definition 4.4. *We will sometimes call an ODE which is linear times a $1/t$ factor, such as $x'(t) = \frac{1}{t} \mathbf{A}x(t) + \frac{1}{t} \mathbf{B}u(t)$, a **linear scale-invariant (LSI)** system.*

4.3.1 Timescale Robustness

As the window size of LegS is adaptive, projection onto this measure is intuitively robust to timescales. Formally, the HIPPO-LegS operator is *timescale-equivariant*: dilating the input f does not change the approximation coefficients.

Lemma 4.5. *For any scalar $\alpha > 0$, if $h(t) = f(\alpha t)$, then $\text{hippo}(h)(t) = \text{hippo}(f)(\alpha t)$. In other words, if $\gamma : t \mapsto \alpha t$ is any dilation function, then $\text{hippo}(f \circ \gamma) = \text{hippo}(f) \circ \gamma$.*

Informally, this is reflected by HIPPO-LegS having *no timescale hyperparameters*; in particular, the discrete recurrence [\(4.3\)](#) is invariant to the discretization step size. By contrast, LegT has a hyperparameter θ for the window size, as well as a step size hyperparameter Δ in the discrete-time case. This hyperparameter is important empirically; Section [4.2.5](#) showed that Δ relates to the gates of RNNs, which are known to be sensitive to their parameterization [99, 198, 72]. We empirically demonstrate the benefits of timescale robustness in Section [4.4.3](#).

Remark 4.3. *Because [\(4.2\)](#) is not LTI, the discretization rules we discussed in Section [2.3.1](#)*

do not directly apply. (4.3) is based on the Euler method for illustration, but HIPPO-LegS is similarly invariant to other discretizations, a property of LSI system. We derive a custom version of the bilinear discretization for LSI dynamics in Appendix A.1, which is what we use instead of (4.3) in practice.

4.3.2 Computational Efficiency

In order to compute a single step of the discrete HIPPO update, the main operation is multiplication by the (discretized) square matrix A . More general discretization specifically requires fast multiplication for any matrix of the form $\mathbf{I} + \Delta \cdot \mathbf{A}$ and $(\mathbf{I} - \Delta \cdot \mathbf{A})^{-1}$ for arbitrary step sizes Δ . Although this is generically a $O(N^2)$ operation, LegS operators use a fixed \mathbf{A} matrix with special structure that turns out to have fast multiplication algorithms for any discretization.¹

Lemma 4.6. *Under any generalized bilinear transform discretization (Appendix A.1), each step of the HIPPO-LegS recurrence in equation (4.3) can be computed in $O(N)$ operations.*

Remark 4.4. *A more general recurrent efficiency result for general HIPPO operators is presented in Chapter 6.*

Section 4.4.1 validates the efficiency of HIPPO layers in practice, where unrolling the discretized versions of Theorem 4.3 is 10x faster than standard matrix multiplication as done in standard RNNs.

4.3.3 Gradient Flow

Much effort has been spent to alleviate the *vanishing gradient problem* in RNNs [156], where backpropagation-based learning is hindered by gradient magnitudes decaying exponentially in time. As LegS is designed for memory, it avoids the vanishing gradient issue.

Lemma 4.7. *For any times $t_0 < t_1$, the gradient norm of HIPPO-LegS operator for the output at time t_1 with respect to input at time t_0 is $\left\| \frac{\partial c(t_1)}{\partial f(t_0)} \right\| = \Theta(1/t_1)$.*

4.3.4 Approximation Error Bounds

The error rate of LegS decreases with the smoothness of the input.

Proposition 4.8. *Let $f: \mathbb{R}_+ \rightarrow \mathbb{R}$ be a differentiable function, and let $g^{(t)} = \text{proj}_t(f)$ be its projection at time t by HIPPO-LegS with maximum polynomial degree $N - 1$. If f*

¹It is known that large families of structured matrices related to orthogonal polynomials are efficient [41].

is L -Lipschitz then $\|f_{\leq t} - g^{(t)}\| = O(tL/\sqrt{N})$. If f has order- k bounded derivatives then $\|f_{\leq t} - g^{(t)}\| = O(t^k N^{-k+1/2})$.

4.4 Empirical Validation

We validate the theory of HIPPO, including its function approximation interpretation and the theoretical results in Section 4.3. Details of all experiments can be found in Appendix D.4.

4.4.1 Online Function Approximation and Speed Benchmark

We empirically show that HIPPO-LegS can scale to capture dependencies across millions of time steps, and its memory updates are computationally efficient (processing up to 470,000 time steps/s) by using an efficient implementation of Lemma 4.6.

Remark 4.5. More visualizations of the HIPPO-LegS and HIPPO-LegT methods are also shown in Figure 5.2 when we revisit HIPPO in Chapter 5.

Long-range Function Approximation

We test the ability of different memory mechanisms in approximating an input function, as described in the problem setup in Section 4.2.1. The model only consists of the memory update (Section 4.3) and not the additional RNN architecture. We choose random samples from a continuous-time band-limited white noise process, with length 10^6 . The model is to traverse the input sequence, and then asked to reconstruct the input, while maintaining no more than 256 units in memory (Figure 4.2). This is a difficult task; the LSTM fails with even sequences of length 1000 ($MSE \approx 0.25$). As shown in Table 4.1, both the LMU and HIPPO-LegS are able to accurately reconstruct the input function, validating that HIPPO can solve the function approximation problem even for very long sequences. Figure 4.2 illustrates the function and its approximations, with HIPPO-LegS almost matching the input function while LSTM unable to do so.

Speed

HIPPO-LegS operator is computationally efficient both in theory (Section 4.3) and in practice. We implement the fast update in C++ with a PyTorch [157] binding and show in Table 4.1 that it can perform 470,000 time step updates per second on a single CPU core, 10x faster than the LSTM and LMU.

Remark 4.6. The LMU showed that its \mathbf{A} matrix (i.e. the HIPPO-LegT matrix) has fast matrix-vector multiplication [218]. However, this only implies a fast recurrence for the Euler-discretized system, but not with the more numerically accurate methods such as bilinear and ZOH (Equation (2.4)) that are necessary in practice; bilinear is used for this task. We note that this matrix does have fast MVM for its bilinear discretization, discussed later in Section 6.1.

Method	Error (MSE)	Speed (elements / s)
LSTM	0.25	35,000
LMU (naive)	0.05	41,000
HIPPO-LegS	0.02	470,000

Table 4.1: Function approximation error after 1 million time steps, with 256 hidden units.

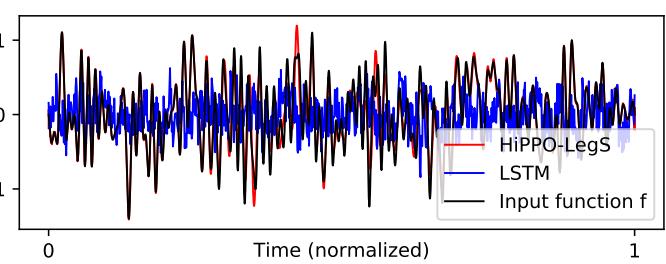


Figure 4.2: Input function and its reconstructions.

4.4.2 Additional Analysis and Ablations of HIPPO Methods

To analyze the tradeoffs of the memory updates derived from our framework, in Figure 4.3 we plot a simple input function $f(x) = 1/4 \sin x + 1/2 \sin(x/3) + \sin(x/7)$ to be approximated. The function is subsampled on the range $x \in [0, 100]$, creating a sequence of length 1000. This function is simpler than the functions sampled from white noise signals described in Section 4.4.1. Given this function, we use the same methodology as in Section 4.4.1 for processing the function online and then reconstructing it at the end.

In Figure 4.3(a, b), we plot the true function f , and its absolute approximation error based on LegT and LegS. LegS has the lowest approximation error, while LegT and LagT are similar and slightly worse than LegS. Next, we analyze some qualitative behaviors.

LegT Window Length

In Figure 4.3(c), shows that the approximation error of LegT is sensitive to the hyperparameter θ , the length of the window. Specifying θ to be even slightly too small (by 0.5% relative to the total sequence length) causes huge errors in approximation. This is expected by the HIPPO framework, as the final measure $\mu^{(t)}$ is not supported everywhere, so the projection problem does not care that the reconstructed function is highly inaccurate near $x = 0$.

LegS vs. LegT

In comparison to LegT, LegS does not need any hyperparameters governing the timescale. However, suppose that the LegT θ window size was chosen perfectly to match the length of the sequence; that is, $\theta = T$ where T is the final time range. Note that at the end of consuming the input function (time $t = T$), the measures $\mu^{(t)}$ for LegS and LegT are *both* equal to $\frac{1}{T}\mathbb{I}_{[0,T]}$ (Sections 4.2.3 and 4.3). Therefore, the approximation $\text{proj}_T(f)$ is specifying the same function for both LegS and LegT at time $t = T$. The sole difference is that LegT has an additional approximation term for $f(t - \theta)$ while calculating the update at every time t (see Appendix D.2.1), due to the nature of the sliding rather than scaling window.

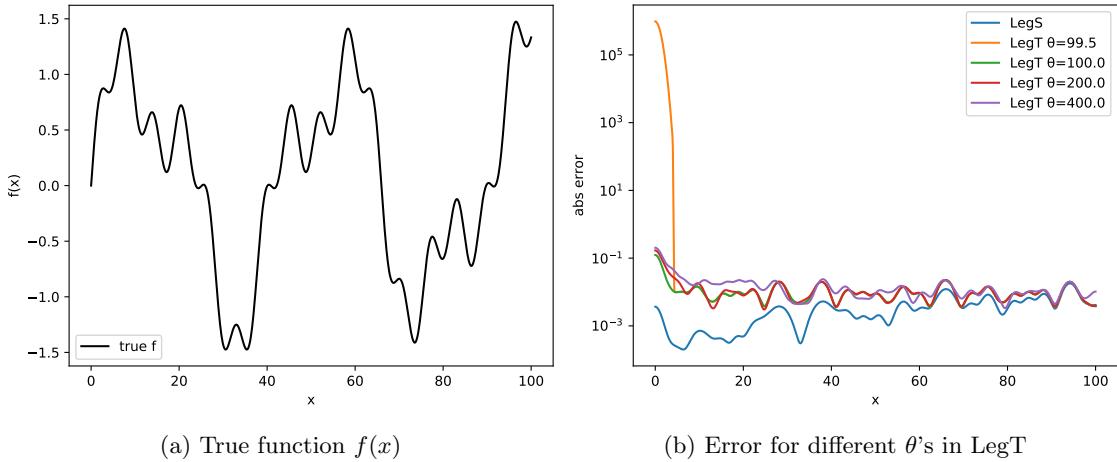


Figure 4.3: Function approximation comparison between LegT and LegS. LegT error is sensitive to the choice of window length θ , especially if θ is smaller than the length of the true function.

Function Approximation Error for Different Discretizations

To understand the impact of approximation error in discretization, in Figure 4.4, we show the absolute error for the HIPPO-LegS updates in function approximation (Section 4.3) for different discretization methods: forward Euler, backward Euler, and bilinear. The bilinear method generally provide sufficiently accurate approximation. We will use bilinear as the discretization method for the LegS updates for the experiments.

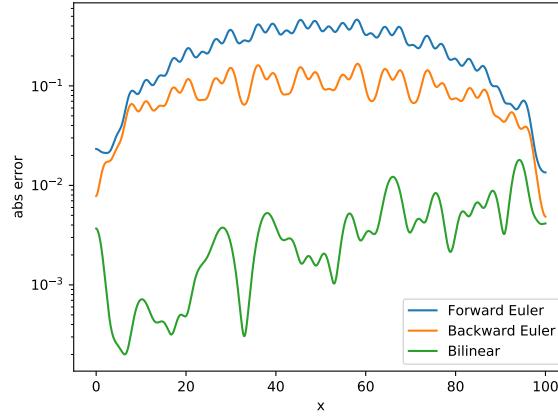


Figure 4.4: Absolute error for different discretization methods. Forward and backward Euler are generally not very accurate, while bilinear yields more accurate approximation.

4.4.3 Timescale Robustness of HIPPO-LegS

Timescale Priors

Sequence models generally benefit from priors on the timescale, which take the form of additional hyperparameters in standard models. Examples include the ‘‘forget bias’’ of LSTMs which needs to be modified to address long-term dependencies [99, 198], or the discretization step size Δ and sliding window length θ of HIPPO-LegT (Section 4.2.4). For example, Section 4.4.2 showed that the sliding window length θ must be set correctly for LegT.

Sampling Rate Change in Trajectory Classification

Recent trends in ML have stressed the importance of understanding robustness under distribution shift, when training and testing distributions are not i.i.d. For time series data, for example, models may be trained on EEG data from one hospital, but deployed at another using instruments with different sampling rates [185, 180]; or a time series may involve the same trajectory evolving at different speeds. Following Kidger et al. [107], we consider the Character Trajectories dataset [9], where the goal is to classify a character from a sequence of pen stroke measurements, collected from one user at a fixed sampling rate. To emulate timescale shift (e.g. testing on another user with slower handwriting), we consider two standard time series generation processes: (1) In the setting of sampling an underlying sequence at a fixed rate, we change the test sampling rate; crucially, the sequences are variable length so the models are unable to detect the sampling rate of the data. (2) In the setting of

irregular-sampled (or missing) data with timestamps, we scale the test timestamps.

Recall that the HIPPO framework models the underlying data as a continuous function and interacts with discrete input only through the discretization. Thus, it seamlessly handles missing or irregularly-sampled data by simply evolving according to the given discretization step sizes (details in Appendix A.1). Combined with LegS timescale invariance (Prop. 4.5), we expect HIPPO-LegS to work automatically in all these settings. We note that the setting of missing data is a topic of independent interest and we compare against SOTA methods, including the GRU-D [23] which learns a decay between observations, and neural ODE methods which models segments between observations with an ODE.

Table 4.2 validates that standard models can go catastrophically wrong when tested on sequences at different timescales than expected. Though all methods achieve near-perfect accuracy ($\geq 95\%$) without distribution shift, aside from HIPPO-LegS, no method is able to generalize to unseen timescales.

Table 4.2: Test set accuracy on Character Trajectory classification on out-of-distribution timescales.

Model	Sampling Rate Change		Missing Values + Timestamps	
	100Hz → 200Hz	200Hz → 100Hz	Upscale	Downscale
LSTM [86]	31.9	28.2	24.4	34.9
GRU [31]	25.4	64.6	28.2	27.3
GRU-D [23]	23.1	25.5	05.5	07.7
ODE-RNN [176]	41.8	31.5	04.3	07.7
NCDE [107]	44.7	11.3	63.9	69.7
LMU [218]	06.0	13.1	39.3	67.8
HIPPO-LegS	88.8	90.1	94.5	94.9

Remark 4.7. *Other forms of handling irregular-sampled and missing data in time series was discussed in Section 2.4.1 and are presented empirically in Section 7.4. A crucial distinction in this setting is that while other continuous-time models can adjust to known changes in the sampling rate, HIPPO-LegS does not even know the changed sampling rate of the data. Handling this requires its scaled measure, which is not present in the time-invariant (LTI) SSMs we use in the rest of this thesis and in related works.*

Remark 4.8. *The results in Table 4.2 required incorporating HIPPO into an RNN architecture where the HIPPO matrices (\mathbf{A}, \mathbf{B}) are frozen and the RNN has other trainable components. This architecture was used in early versions of HIPPO, but is no longer recommended, in favor of incorporating it into a state space model instead. Details of the original architecture can be found in Gu et al. [70].*

Remark 4.9. *Unlike our more general empirical results in Part III, which use special SSM representations (Chapter 3) where all parameters are trainable, the experiments here use frozen (\mathbf{A}, \mathbf{B}) matrices. More visualizations of these methods are also provided when we revisit HIPPO in Chapter 5.*

4.5 Conclusion

We address the fundamental problem of memory in sequential data by proposing a framework (HIPPO) that poses the abstraction of optimal function approximation with respect to time-varying measures. In addition to unifying and explaining existing memory approaches, HIPPO unlocks a new method (HIPPO-LegS) that takes a first step toward timescale robustness and can efficiently handle dependencies across millions of time steps. HIPPO historically predated SSMs for deep sequence modeling (e.g. S4); the next two chapters describe it from an SSM perspective and connect it with Part I.

Chapter 5

HIPPO as Orthogonal SSMs

Chapter 4 introduced the HIPPO framework as a first-principles approach to online memorization with continuous/recurrent methods, without referencing state space models. This chapter revisits the same framework through the lens of SSMs. Our SSM-centric approach refines and generalizes HIPPO through a theoretically rich class of new state space models. We develop more instantiations of HIPPO for other nice measure and basis functions; further, unlike some instantiations in Chapter 4, these methods are all *time-invariant* SSMs that can be used as an SSSM (Remark 2.6). Our framework also enables analyzing several aspects of training this family of SSMs, such as how to initialize the other SSM parameters.

- Section 5.1 revisits the HIPPO framework from the perspective of SSMs, and introduces a notion of orthogonal SSM that captures its main purpose of online memorization.
- In Section 5.2, we derive a new method that takes the HIPPO-LegS matrices from Chapter 4, but uses them in a time-invariant (LTI) instead of time-varying SSM. We show that this is an orthogonal SSM (i.e. is still a HIPPO method) with an exponentially decaying measure (Figure 5.1 (*Left*)).
- In Section 5.3, we focus on the case of finite-window SSMs, deriving a new method and proving previously established conjectures for old methods. The new method HIPPO-FouT is based on *truncated Fourier basis functions*. This method thus automatically captures sliding Fourier transforms (e.g. the STFT and spectrograms), which are ubiquitous as a hand-crafted signal processing tool, and can also represent any *local convolution*, thus generalizing conventional CNNs (Figure 5.1 (*Middle*)).

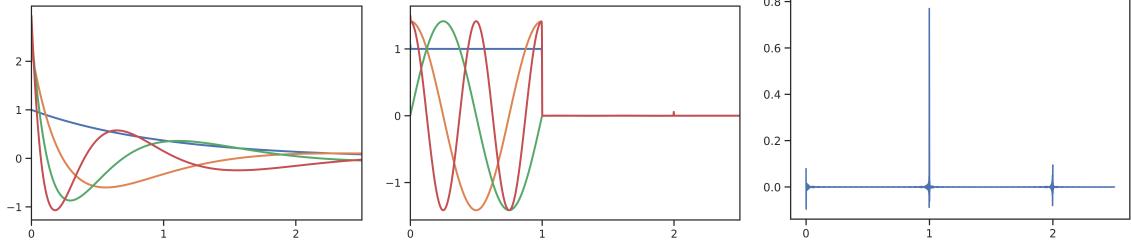


Figure 5.1: This chapter derives several new HIPPO methods. (*Left: LegS*) We prove that using the HIPPO-LegS (\mathbf{A}, \mathbf{B}) matrices as a time-invariant SSM produces Legendre polynomials under an exponential re-scaling, resulting in smooth basis functions with a closed form formula. (*Middle, Right: FouT*) We define a new SSM that produces approximations to the **truncated Fourier** basis, perhaps the most intuitive and ubiquitous set of basis functions. This method generalizes sliding Fourier Transforms and local convolutions (i.e. CNNs), and can also encode spike functions to solve classic memorization tasks.

- Section 5.4 shows more general properties of TO-SSMs, which establish guidelines for interpreting and initializing the other SSM parameters. In particular, we provide an intuitive explanation of the timescale Δ (Section 2.4.1), which has a precise interpretation as controlling the length of dependencies that the model captures. We also show how to initialize the last parameter \mathbf{C} to make an SS(S)M layer variance-preserving and stable.

5.1 Framework: Revisiting HIPPO as Orthogonal SSMs

Remark 5.1. *We recommend revisiting the notation and definitions in Chapter 2, in particular Definition 2.6, Definition 2.7, and Definition 2.18.*

5.1.1 Summary of HIPPO Matrices

HIPPO produced several operators with particular formulas. For example, the LegS method was defined as $x'(t) = \frac{1}{t}\mathbf{A}x(t) + \frac{1}{t}\mathbf{B}u(t)$ for (\mathbf{A}, \mathbf{B}) given by (5.1). Similarly, the LegT method is $x'(t) = \mathbf{A}x(t) + \mathbf{B}u(t)$ for equation (5.2). We also define a third formula FouT that we will introduce in this chapter (5.3).

(HIPPO-LegS)

$$\mathbf{A}_{nk} = -(2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} \cdot \begin{cases} 1 & n > k \\ \frac{n+1}{2n+1} & n = k \\ 0 & n < k \end{cases} \quad (5.1)$$

$$\mathbf{B}_n = (2n+1)^{\frac{1}{2}}$$

(HIPPO-FouT)

$$\mathbf{A}_{nk} = \begin{cases} -2 & n = k = 0 \\ -2\sqrt{2} & n = 0, k \text{ odd} \\ -2\sqrt{2} & k = 0, n \text{ odd} \\ -4 & n, k \text{ odd} \\ 2\pi k & n - k = 1, k \text{ odd} \\ -2\pi n & k - n = 1, n \text{ odd} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{B}_n = \begin{cases} 2 & n = 0 \\ 2\sqrt{2} & n \text{ odd} \\ 0 & \text{otherwise} \end{cases} \quad (5.3)$$

(HIPPO-LegT)

$$\mathbf{A}_{nk} = -(2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} \cdot \begin{cases} 1 & n \geq k \\ (-1)^{n-k} & n \leq k \end{cases}$$

$$\mathbf{B}_n = (2n+1)^{\frac{1}{2}} \quad (5.2)$$

These matrices were originally motivated by the question of *online memorization* of an input signal. We now present an improved framework that addresses this problem through an SSM-centric approach. In particular, we define new subclasses of SSMs with special properties that can be used for online function reconstruction, simplifying and generalizing the original HIPPO framework.

First, we observe that the HIPPO formulas are obviously SSMs by Definition 2.6.

Remark 5.2. *In this chapter, we do not assume LTI SSMs (Section 2.2.1). More formally, the HIPPO-LegS in Chapter 4 is an SSM $(\mathbf{A}(t), \mathbf{B}(t))$ defined as $(\frac{1}{t}\mathbf{A}, \frac{1}{t}\mathbf{B})$ with (5.1).*

The first step is to rephrase the definition of HIPPO operators in the language of SSMs. The goal of HIPPO is then to find a suitably chosen SSM basis (\mathbf{A}, \mathbf{B}) , so that at any time t , the current state $x(t)$ can be used to approximately reconstruct the entire input u up to time t (Figure 5.2).

5.1.2 Orthogonal State Space Models

Our key abstraction is the following definition.

Definition 5.1. *We call an SSM $(\mathbf{A}(t), \mathbf{B}(t))$ an **orthogonal SSM (O-SSM)** for the basis $p_n(t, s)$ and measure $\omega(t, s) \geq 0$ if the functions $K_n(t, s) = p_n(t, s)\omega(t, s)$ satisfy, at*

all times t ,

$$x_n(t) = \int_{-\infty}^t K_n(t, s) u(s) ds \quad \int_{-\infty}^t p_n(t, s) p_m(t, s) \omega(t, s) ds = \delta_{n,m}. \quad (5.4)$$

In the case of a **time-invariant O-SSM (TO-SSM)**, $K_n(t, s) =: K_n(t-s)$ (depends only on $t-s$), giving us Definition 2.18 with measure $\omega(t-s) := \omega(t, s)$ and basis $p_n(t-s) := p_n(t, s)$.

To be more specific about terminology, p_n and ω are called the *basis and measure* for *orthogonal SSMs* (Definition 5.1), while K_n are called the *SSM basis kernels* which applies more generally to all SSMs (Definition 2.18). The distinction will be made clear from context, notation, and the word “kernel” referring to K_n .

For O-SSMs, (p, ω) and K are uniquely determined by each other, so we can refer to an O-SSM by either. One direction is obvious: (p, ω) determine K via $K_n(t, s) = p_n(t, s)\omega(t, s)$.

Proposition 5.2. *If a set of kernel functions satisfies $K_n(t, s) = p_n(t, s)\omega(t, s)$ where the functions p_n are complete and orthogonal w.r.t. ω (equation (5.4) right), p and ω are unique.*

Equation (5.4) is equivalent to saying that for every fixed t , $\langle p_n, p_m \rangle_\omega = \delta_{n,m}$, or that p_n are an orthonormal basis with respect to measure ω . More formally, defining $p_n^{(t)}(s) = p_n(t, s)$ and $\omega^{(t)}$ similarly, then $p_n^{(t)}$ are orthonormal in the Hilbert space with inner product $\langle p, q \rangle = \int p(s)q(s)\omega^{(t)}(s) ds$. By equation (5.4), $x_n(t) = \int_{-\infty}^t u(s)K_n(t, s) ds = \langle u, p_n^{(t)} \rangle_{\omega^{(t)}}$ where $p_n^{(t)}(s) = p_n(t, s)$. Thus at all times t , the state vector $x(t)$ is simply *the projections of $u|_{\leq t}$ onto a orthonormal basis*, so that the history of u can be reconstructed from $x(t)$. HIPPO calls this the **online function approximation** problem (Chapter 4).

Proposition 5.3. *Consider an O-SSM that satisfies (5.4) and fix a time t . Furthermore suppose that in the limit $N \rightarrow \infty$, the $p_n^{(t)}$ are a complete basis on the support of $\omega^{(t)}$. Then $u(s) = \sum_{n=0}^{\infty} x_n(t)p_n(t, s)$ for all $s \leq t$.*

The main barrier to using Proposition 5.3 for function reconstruction is that SSMs are in general not O-SSMs. For example, even though we will show that (5.1) is an TO-SSM, and that unitary conjugation of a TO-SSM is a TO-SSM (Section 5.4), its diagonal matrix of eigenvalues is not a TO-SSM. This both shows the existence of an SSM that is not an O-SSM, and also implies that general conjugation does not preserve TO-SSMs.

Proposition 5.4. *There is no TO-SSM with the diagonal state matrix $A = \text{diag}\{-1, -2, \dots\}$.*

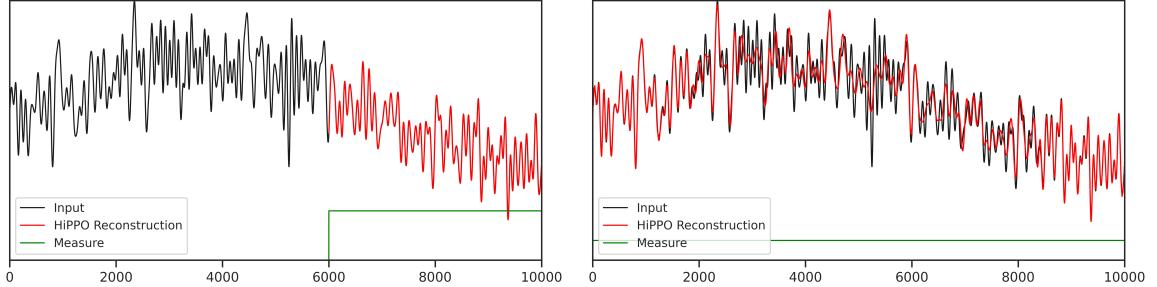


Figure 5.2: (**Prior HIPPO methods.**) Given an input function $u(t)$ (black), HIPPO compresses it online into a state vector $x(t) \in \mathbb{R}^N$ via equation (2.1a). In Chapter 4, specific cases of HIPPO matrices \mathbf{A}, \mathbf{B} are derived so that at every time t , the history of u up to time t can be reconstructed linearly from $x(t)$ (red), according to a measure (green). (*Left*) The HIPPO-LegT method orthogonalizes onto the Legendre polynomials against a time-invariant uniform measure, i.e. sliding windows. (*Right*) The original HIPPO-LegS method is *not* a time-invariant system. When used as a time-varying ODE $x' = \frac{1}{t}\mathbf{A}x + \frac{1}{t}\mathbf{B}u$, $x(t)$ represents the projection of the entire history of u onto the Legendre polynomials.

HIPPO can thus be viewed as a framework for deriving specific SSMs that *do* satisfy (5.4). The HIPPO methods in Chapter 4 primarily focused on the case when the p_n are orthogonal polynomials, and specifically looked for solutions to (5.4), which turn out to be SSMs. We have rephrased the HIPPO definition in Definition 5.1 to start directly from SSMs and hence is more general. We now revisit the two most important cases previously introduced.

5.1.3 Previous HIPPO Methods

HIPPO-LegT

Chapter 4 showed that (\mathbf{A}, \mathbf{B}) given by (5.2) is a TO-SSM that approximates the truncated Legendre polynomials (Figure 5.3).

Definition 5.5. Let $\mathbb{I}(t)$ be the indicator function for the unit interval $[0, 1]$. Let $L_n(t)$ be the Legendre polynomials rescaled to be orthonormal on $[0, 1]$, i.e., $\int L_n(t)L_m(t)\mathbb{I}(t) dt = \delta_{n,m}$.

Proposition 5.6. As $N \rightarrow \infty$, the SSM with (\mathbf{A}, \mathbf{B}) in (5.2) is a TO-SSM with

$$\omega(t) = \mathbb{I}(t) \quad K_n(t) = L_n(t)\mathbb{I}(t).$$

This particular system was the precursor to HIPPO and has also been variously called the Legendre Delay Network (LDN) or Legendre Memory Unit (LMU) [219, 218]. The original motivation of this system was not through the online function approximation formulation

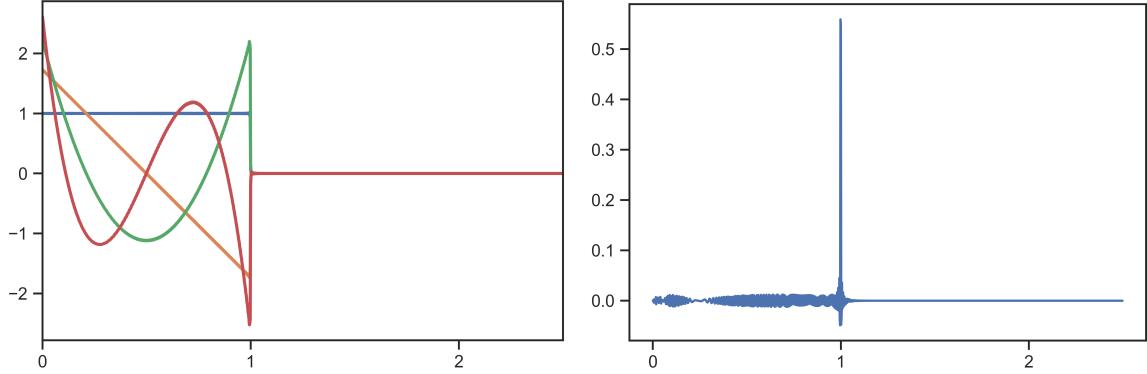


Figure 5.3: (**LegT.**) (*Left*) First 4 basis functions $K_n(t)$ for state size $N = 1024$ (Proposition 5.6). (*Right*) Choosing a particular \mathbf{C} produces a spike kernel or “delay network” (Theorem 5.14).

of HIPPO, but through finding an optimal SSM approximation to the **delay network** that has impulse response $K(t) = \delta(t - 1)$ representing a time-lagged output by 1 time unit (Figure 5.3). We state and provide an alternate proof of this result in Appendix E.4.4, Theorem 5.14.

HIPPO-LegS

Unlike the HIPPO-LegT case, which is an LTI system (2.1a) (i.e. TO-SSM), the HIPPO-LegS matrix (5.1) from Chapter 4 was *meant to be used in a time-varying system* $x'(t) = \frac{1}{t} \mathbf{A}x(t) + \frac{1}{t} \mathbf{B}u(t)$. In contrast to HIPPO-LegT, which reconstructs onto the truncated Legendre polynomials in sliding windows $[t - 1, t]$, HIPPO-LegS reconstructs onto Legendre polynomials on “scaled” windows $[0, t]$; since the window changes across time, the system is not time-invariant (Figure 5.2). Specifically, we have:

Theorem 5.7. *The SSM $(\frac{1}{t}\mathbf{A}, \frac{1}{t}\mathbf{B})$ for (\mathbf{A}, \mathbf{B}) in (5.1) is an O-SSM with*

$$\omega(t, s) = \frac{1}{t} \cdot \mathbb{I}(s/t) \quad p_n(t, s) = L_n(s/t).$$

Surprisingly, these particular (\mathbf{A}, \mathbf{B}) matrices are very special, and can be flexibly transformed in several ways while still being an O-SSM. As the most important case of this phenomenon, it turns out that the (time-invariant) SSM (\mathbf{A}, \mathbf{B}) is still an O-SSM, and thus a TO-SSM. This will turn out to be the most important case of a T-SSM that can be used as a sequence model (recall that we required SSSMs to be T-SSMs, Remark 2.6).

5.1.4 The General Result

Our fully general result is Theorem E.3 in Appendix E.2, which describes a very general way to derive O-SSMs for various SSM basis functions $K_n(t, s)$. This result can be instantiated in many ways to generalize all previous results in this line of work. We do not state it in full detail here, but show how it can be used to recover these cases.

5.2 Generalizations of LegS

We showcase the generality of Theorem E.3 by stating the following special case containing a subclass of time-varying O-SSMs (which are themselves rich enough to cover both time-invariant and time-varying versions of HIPPO-LegS):

Corollary 5.8. *Define $\sigma(t, s) = \exp(a(s) - a(t))$ for any differentiable function a . The SSM $(a'(t)\mathbf{A}, a'(t)\mathbf{B})$ is an O-SSM with*

$$\omega(t, s) = \mathbb{I}(\sigma(t, s))a'(s)\sigma(t, s) \quad p_n(t, s) = L_n(\sigma(t, s)).$$

We show the matrices (\mathbf{A}, \mathbf{B}) in (5.1) are deeply related to the Legendre polynomials L_n defined in Definition 5.5. In particular, as more specific corollaries of Corollary 5.8, we recover both the original time-varying interpretation of the matrix in (5.1), as well as the instantiation of LegS as a time-invariant system. If we set $a'(t) = \frac{1}{t}$, then we recover the scale-invariant HIPPO-LegS O-SSM in Theorem 5.7:

Corollary 5.9 (Scale-Invariant HIPPO-LegS, Theorem 5.7). *The SSM $(\frac{1}{t}\mathbf{A}, \frac{1}{t}\mathbf{B})$ is a TO-SSM for basis functions $K_n(t) = \frac{s}{t}L_n(\frac{s}{t})$ and measure $\omega = \frac{1}{t}\mathbb{I}[0, 1]$ where \mathbf{A} and \mathbf{B} are defined as in (5.1).*

And if we set $a'(t) = 1$, this shows that time-invariant HIPPO-LegS is a TO-SSM:

Corollary 5.10 (Time-Invariant HIPPO-LegS). *The SSM (\mathbf{A}, \mathbf{B}) is a TO-SSM with*

$$\omega(t) = e^{-t} \quad p_n(t) = L_n(e^{-t}).$$

In brief, this is orthogonalizing onto the Legendre polynomials with an exponential “warping” or change of basis on the time axis (Figure 5.1,(Left)).

5.3 Finite Window TO-SSMs

For the remainder of this section, we restrict to the time-invariant SSM setting (Section 2.2.1). A second important instantiation of Theorem E.3 covers cases with a discontinuity in the SSM basis functions $K_n(t)$, which requires infinite-dimensional SSMs to represent. The most important type of discontinuity occurs when $K_n(t)$ is supported on a finite window, so that these T-SSMs represent sliding window transforms.

We first derive a new sliding window transform based on the widely used Fourier basis (Section 5.3.1). We also prove results relating finite window methods to delay networks (Section 5.3.2).

5.3.1 HIPPO-FouT

Using the more general framework (Theorem E.3) that does not necessarily require polynomials as basis functions, we derive a TO-SSM that projects onto **truncated Fourier functions**.

Theorem 5.11. *As $N \rightarrow \infty$, the SSM for (5.3) is a TO-SSM with $\omega(t) = \mathbb{I}(t)$, and $\{p_n\}_{n \geq 1}$ are the truncated Fourier basis functions orthonormal on $[0, 1]$, ordered in the form $\{p_n\}_{n \geq 0} = (1, c_0(t), s_0(t), \dots)$, where $s_m(t) = \sqrt{2} \sin(2\pi m t)$ and $c_m(t) = \sqrt{2} \cos(2\pi m t)$ for $m = 0, \dots, N/2$.*

This SSM corresponds to Fourier series decompositions, a ubiquitous tool in signal processing, but represented as a state space model. The basis is visualized in Figure 5.1 (middle) for state size $N = 1024$.

A benefit of using these well-behaved basis functions is that we can leverage classic results from Fourier analysis. For example, it is clear that taking linear combinations of the truncated Fourier basis can represent any function on $[0, 1]$, and thus FouT can represent any local convolution (i.e. the layers of modern convolutional neural networks).

Theorem 5.12. *Let $K(t)$ be a differentiable kernel on $[0, 1]$, and let $\hat{K}(t)$ be its representation by the FouT system (Theorem 5.11) with state size N . If K is L -Lipschitz, then for $\epsilon > 0$, $N \geq (\frac{L}{\pi\epsilon})^2 + 2$, we have $\|K(t) - \hat{K}(t)\| \leq \epsilon$. If K has k -derivatives bounded by L , then we can take $N \geq (\frac{L}{\pi^k\epsilon})^{\frac{2}{2k-1}} + 2$.*

5.3.2 Approximating Delay Networks

An interesting property of these finite window T-SSMs is that they can approximate **delay functions**. This is defined as a system with impulse response $K(t) = \delta(t - 1)$: then $y(t) = (K * u)(t) = u(t - 1)$, which means the SSM outputs a time-lagged version of the input. This capability is intuitively linked to HIPPO, since in order to do this, the system must be remembering the entire window $u([t - 1, t])$ at all times t , in other words perform an *approximate function reconstruction*. Any HIPPO method involving finite windows should have this capability, in particular, the finite window methods LegT and FouT.

Theorem 5.13. *For the FouT system (\mathbf{A}, \mathbf{B}) , let \mathbf{C} be (twice) the vector of evaluations of the basis functions $\mathbf{C}_n = 2 \cdot p_n(1)$ and let $\mathbf{D} = 1$. For the LegT system (\mathbf{A}, \mathbf{B}) , let \mathbf{C} be the vector of evaluations of the basis functions $\mathbf{C}_n = p_n(1) = (1 + 2n)^{\frac{1}{2}}(-1)^n$ and let $\mathbf{D} = 0$.*

In both cases, the SSM kernel $K(t) = \mathbf{C}e^{t\mathbf{A}}\mathbf{B} + \mathbf{D}\delta(t)$ limits to $K(t) \rightarrow \delta(t - 1)$ as $N \rightarrow \infty$.

Theorem 5.13 is visualized in Figures 5.1 and 5.3 (right). Further, the result for LegT can be characterized even more tightly for finite N . In fact, this was the original motivation for the LDN/LMU [219, 218], which worked backward from the transfer function of the desired delay function impulse response $K(t) = \delta(t - 1)$, and noticed that the SSM for Padé approximations to this were linked to Legendre polynomials. This was not fully proven, and we state it here and provide a full proof in Appendix E.4.

Theorem 5.14. *For $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ in the LegT system described in Theorem 5.13, the transfer function $\mathcal{L}\{K(t)\}(s)$ is the $[N - 1/N]$ Padé approximant to $e^{-s} = \mathcal{L}\{\delta(t - 1)\}(s)$.*

We remark that although LegT (LMU) is designed to be an “optimal” approximation to the delay function via Padé approximants, it actually produces a weaker spike function than FouT (Figure 5.3 vs. Figure 5.1) and empirically performs slightly worse on synthetic tasks testing this ability (Section 7.5.8). This may be because Padé approximation in the Laplace domain does not necessarily translate to localization in the time domain.

The above result provides theoretical justification for why S4-FouT excels at dense memorization tasks (see Section 7.5.8).

5.4 Properties of TO-SSMs: Timescales and Normalization

We describe several theoretical properties of TO-SSMs, which answer questions such as:

- How should all parameters $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ be initialized for an SS(S)M layer to be properly normalized?
- What does Δ intuitively represent, and how should it be set in an SSM model?

While these might not be clear for general SSMs, it turns out that for TO-SSMs, these two questions are closely related and have intuitive interpretations.

Closure properties. First, several basic transformations preserve the structure of TO-SSMs. Consider a TO-SSM (\mathbf{A}, \mathbf{B}) with basis functions $p_n(t)$ and measure $\omega(t)$. Then, for any scalar c and unitary matrix \mathbf{V} , the following are also TO-SSMs with the corresponding basis functions and measure (Appendix E.5, Lemma E.15):

Transformation	Matrices	Interpretation	Basis	Measure
Scalar Scaling	$(c\mathbf{A}, c\mathbf{B})$	Timescale change	$p(ct)$	$\omega(ct)c$
Identity Shift	$(\mathbf{A} + c\mathbf{I}, \mathbf{B})$	Exponential tilting	$p(t)e^{-ct}$	$\omega(t)e^{2ct}$
Unitary Transformation	$(\mathbf{V}\mathbf{A}\mathbf{V}^*, \mathbf{V}\mathbf{B})$	Identity	$\mathbf{V}p(t)$	$\omega(t)$

Normalization. A standard aspect of training deep learning models, in general, concerns the scale or variance of activations. This has been the subject of much research on training deep learning models, touching on deep learning theory for the dynamics of training such as the exploding/vanishing gradient problem [85], and a large number of normalization methods to ensure properly normalized methods, from the simple Xavier/He initializations [66, 80] to BatchNorm and LayerNorm [92, 7] to many modern variants and analyses of these [39].

The following proposition follows because for a TO-SSM, $x(t)$ can be interpreted as projecting onto orthonormal functions in a Hilbert space (Proposition 5.3).

Lemma 5.15 (Normalization of TO-SSM). *Consider an (infinite-dimensional) TO-SSM. For any input $u(t)$, $\|x(t)\|_2^2 = \|u\|_\omega^2 = \int_{-\infty}^t u(s)^2 \omega(t-s) dt$.*

Corollary 5.16. *For a TO-SSM with a probability measure (i.e. $\int \omega(t) = 1$) and any constant input $u(t) = c$, the state has norm $\|x(t)\|^2 = c^2$ and the output $y(t)$ has mean 0, variance c^2 if the entries of \mathbf{C} are mean 0 and variance 1.*

Note that the probability measure requirement can be satisfied by simply rescaling \mathbf{B} . Corollary 5.16 says the TO-SSM preserves the variance of inputs, the critical condition for a properly normalized deep learning layer. Note that the initialization of \mathbf{C} is different than

a standard **Linear** layer in deep neural networks, which usually rescale by factor depending on its dimensionality such as $N^{-\frac{1}{2}}$ [66].

Timescales. As discussed in Sections 2.3.1 and 2.4.1, converting from continuous to discrete time involves a parameter Δ that represents the step size of the discretization. This is an unintuitive quantity when working directly with discrete data, especially if it is not sampled from an underlying continuous process.

By Lemma 2.13, however, Δ can be viewed just as a scalar scaling of the base SSM instead of changing the rate of the input. In the context of TO-SSMs, this just stretches the underlying basis and measure (Appendix E.5, Scalar Scaling).

The most intuitive example of this is for a finite window TO-SSM such as LegT or FouT. Discretizing this system with step size Δ is equivalent to considering the system $(\Delta \mathbf{A}, \Delta \mathbf{B})$ with step size 1, which produces basis functions supported exactly on $[0, \frac{1}{\Delta}]$. The interpretation of the timescale Δ lends to simple discrete-time corollaries of the previous continuous-time results. For example, LegT and FouT *represent sliding windows of $1/\Delta$ elements* in discrete time.

Corollary 5.17. *By Theorem 5.13, as $N \rightarrow \infty$, the discrete convolutional kernel $\bar{\mathbf{K}} \rightarrow e_{\lceil \Delta^{-1} \rceil}$, i.e. the discrete delay network with lag $\frac{1}{\Delta}$.*

Corollary 5.18. *For HIPPO-FouT matrices (\mathbf{A}, \mathbf{B}) , by Theorem 5.11, as $N \rightarrow \infty$, the discrete convolutional kernel $\bar{\mathbf{K}}$ (over the choice of \mathbf{C}) can represent any local convolution of length $\lfloor \Delta^{-1} \rfloor$.*

This discussion motivates the following definition. Properly normalized TO-SSMs (\mathbf{A}, \mathbf{B}) will model dependencies of expected length 1, and Δ modulates it to model dependencies of length $\frac{1}{\Delta}$, allowing fine-grained control of the context size of a TO-SSM.

Definition 5.19 (Timescale of TO-SSM). *Define $\mathbb{E}[\omega] = \frac{\int_0^\infty t\omega(t)dt}{\int_0^\infty \omega(t)dt}$ to be the timescale of a TO-SSM having measure $\omega(t)$. A TO-SSM is timescale-normalized if it has timescale 1.*

By this definition, HIPPO-LegS is timescale-normalized. On the other hand, HIPPO-LegT and -FouT were derived with measures $\mathbb{I}[0, 1]$, which have timescale $\frac{1}{2}$ by Definition 5.19. To properly normalize them, we could actually halve the matrices to make them orthogonal w.r.t. $\omega = \frac{1}{2}\mathbb{I}[0, 2]$.

Table 5.1: Summary of time-invariant orthogonal SSMs (TO-SSMs).

Method	SSM kernels $e^{t\mathbf{A}}\mathbf{B}$	Orthogonal basis $p_n(t)$	Measure $\omega(t)$	Timescale
LegS (5.1)	$L_n(e^{-t})e^{-t}$	$L_n(e^{-t})$	e^{-t} (exponential)	1
FouT (5.3)	$(\cos, \sin)(2\pi nt)\mathbb{I}[0, 1]$	$(\cos, \sin)(2\pi nt)$	$\mathbb{I}[0, 1]$ (uniform)	$1/2$
LegT (5.2)	$L_n(t)\mathbb{I}[0, 1]$	$L_n(t)$	$\mathbb{I}[0, 1]$ (uniform)	$1/2$

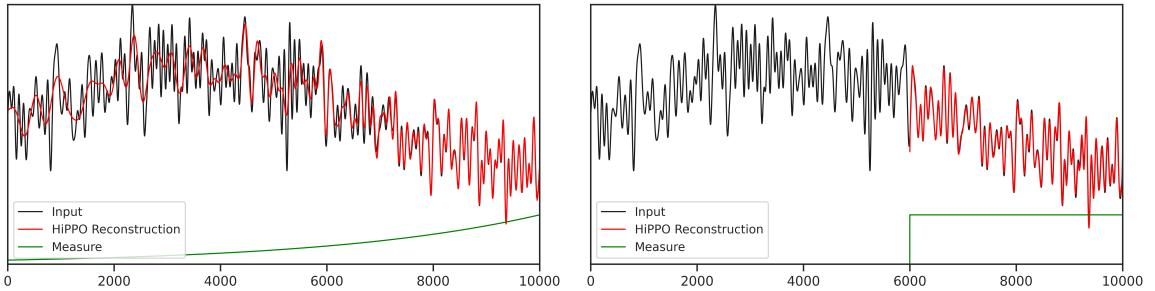


Figure 5.4: (New HIPPO methods.) Function reconstruction predicted by our general theory. An input signal of length 10000 is processed sequentially, maintaining a state vector of size only $x(t) \in \mathbb{R}^{64}$, which is then used to approximately reconstruct the entire history of the input. (Left) LegS (as an LTI system) orthogonalizes on the Legendre polynomials warped by an exponential change of basis, smoothening them out. This basis is orthogonal with respect to an exponentially decaying measure. Matching the intuition, the reconstruction is very accurate for the recent past and degrades further out, but still maintains information about the full history of the input, endowing it with long-range modeling capacity. (Right) FouT orthogonalizes on the truncated Fourier basis, similar to the original HIPPO-LegT or LMU.

5.5 Summary and Discussion

This chapter recasts the HIPPO framework for online memorization into the language of SSMs, while generalizing it to any set of orthonormal basis functions as projection operators. In this language, HIPPO is a general mathematical framework for producing matrices (\mathbf{A}, \mathbf{B}) corresponding to prescribed families of well-behaved basis functions. Our new orthogonal SSM abstraction leads to new methods for different basis functions, and principled explanations of other components such as the timescale and initialization.

Table 5.1 summarizes the special cases of TO-SSMs presented in this chapter.

5.5.1 New Methods and Visualizations

Figure 5.4 confirms the HIPPO theory of online function reconstruction (Proposition 5.3) for the proposed methods LegS and FouT. (Contrast to Figure 5.2, which visualizes the methods introduced in Chapter 4.) These are TO-SSMs corresponding to exponentially-scaled

Legendre families (LegS) and the truncated Fourier functions (FouT). Intuitively,

- LegS produces a very smooth, long-range family of kernels (Figure 5.1). Empirically, it is usually our best method for long-range dependencies among all S4 variants.
- FouT is a finite window method that generalizes local convolutions and captures important transforms such as the sliding DFT or STFT.

5.5.2 General HIPPO Operators

We focused on specific cases with nice measures and basis functions, such as Fourier and Legendre bases. Our framework is actually much more general, and implies that there exist O-SSMs corresponding to all families of orthogonal polynomials $\{p_n(t)\}$. In other words, all orthogonal polynomials can be defined as the SSM kernels of some (\mathbf{A}, \mathbf{B}) . These follow as corollaries of our general results (Appendix E.2).

Remark 5.3. *The HIPPO acronym is somewhat of a misnomer due to historical reasons. It is named for polynomials because these were the only methods produced by the original paper. However the core concept is more general; even the original definition of HIPPO operators in Definition 4.1, which was specialized to polynomial bases, can be trivially re-defined to use any basis functions instead of polynomials.*

Remark 5.4. *The difficulty of using non-polynomial bases essentially reduces to the following. The crucial property that makes HIPPO work is that the basis functions p_n must have derivatives that can be expressed as linear combinations of the other basis functions. Polynomials are a natural choice, and it can be difficult to find other families that have this property and are also complete. However some do exist, such as the Fourier basis that leads to FouT.*

If we were to give a name to the special case that HIPPO originally considered, they would be polynomial O-SSMs (POSSMs). Therefore, a central result of our theory is that all HIPPOs are POSSMs.

5.5.3 Interpretation and Initialization of TO-SSMs as Sequence Models

All methods described have produced fixed formulas for (\mathbf{A}, \mathbf{B}) . Recall from Definition 2.18 that we can view a full SSM $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ as representing convolution kernels that are linear combinations (parameterized by \mathbf{C}) of basis functions (parameterized by (\mathbf{A}, \mathbf{B})). Thus, all

HIPPO methods we have derived can be interpreted as specifying a fixed set of nice basis functions, which can be incorporated into a downstream sequence model (SSSM) which learns the linear combination \mathbf{C} and kernel width Δ .

The theory of TO-SSMs also describes how to properly set these other SSSM parameters. Consider a timescale-normalized orthogonal SSM (\mathbf{A}, \mathbf{B}) , i.e. $\int_0^\infty \omega(t) = 1$ and $\int_0^\infty t\omega(t) = 1$. For a full SSSM using these matrices, the other parameters \mathbf{C} and Δ should be defined as follows.

Remark 5.5 (Initialization of \mathbf{C}). *\mathbf{C} should be initialized with independent isotropic (mean 0, variance 1) entries. This ensures that it is a variance-preserving SSM, i.e. produces outputs y matching the variance of the input u .*

Remark 5.6 (Initialization of Δ). *Independent of the mechanics of discretization (Section 2.3.1), the timescale $\frac{1}{\Delta}$ has a simple interpretation as controlling the length of dependencies the SSM captures, or the width of the SSM kernels. For example, for a finite window measure such as FouT, the kernels have length exactly $\frac{1}{\Delta}$ and can represent any finite convolution kernel of this length (generalizing the standard local convolutions used in deep learning). A sensible initialization is to choose a lower and upper bound $(\Delta_{\min}, \Delta_{\max})$ and initialize Δ log-uniformly randomly in this range. (Recall that we consider SSSMs with H independent heads (Section 2.3.5) and thus have H values of Δ filling this range.) For example, a good default value that works for long-range tasks is $(\Delta_{\min}, \Delta_{\max}) = (0.001, 0.1)$, covering a geometric range of sensible timescales (expected length 10 to 1000). In Section 7.5.8 we show that the timescale can be chosen more precisely when lengths of dependencies are known.*

Chapter 6

Combining Orthogonal and Structured State Space Models

In Chapters 2 and 3, we defined structured SSM models (S4), which combine continuous, recurrent, and convolutional properties, while being very efficient to compute. However, we did not define concrete instantiations of these models that can address modeling challenges such as long-range dependencies (Chapter 1).

On the other hand, in Chapters 4 and 5 we defined SSMs with special mathematical properties through the HIPPO framework. In particular, they address the problem of online memorization as an approach to modeling long-range dependencies (Chapter 4), and can be viewed as orthogonal SSMs that decompose an input onto an orthogonal system of smooth basis functions (Chapter 5). However, we did not discuss how to compute these models.

This chapter combines the two concepts: we show that the most important orthogonal SSMs, corresponding to named HIPPO methods (Section 5.1.1), all can be written as structured SSMs. Consequently, the class of S4 models is simultaneously optimally efficient (Definitions 3.1 and 3.2), and can be instantiated with HIPPO matrices for additional theoretical properties and capabilities such as long-range memory.

- Section 6.1 rehashes the main challenges of, and presents partial solutions to, resolving the computational efficiency problem of SSMs using HIPPO.
- Section 6.2 contains our most important results: the named HIPPO matrices can all

be exactly written as the DPLR matrices from Chapter 3. This allows S4 models to be instantiated using HIPPO methods, resolving their drawbacks.

- Section 6.3 shows further theoretical results that the most important HIPPO matrix can be approximated by a diagonal matrix, allowing S4D to also be instantiated using HIPPO. Other diagonal variants are provided.

6.1 Overview: Motivation and Partial Progress

Structured SSMs are not necessarily effective. The generality of SS(S)Ms means they can inherit the issues of recurrences and convolutions at addressing long dependencies (Section 1.2.3). For example, viewed as a recurrence, repeated multiplication by \bar{A} could suffer from the well-known **vanishing gradients** problem [156, 174]. Empirically, Section 7.5 shows simple ablations confirming that SSMs with random state matrices can be ineffective as a general sequence model.

Principled SSMs are not necessarily efficient. The HIPPO framework describes how to memorize a function in continuous time with respect to a measure ω [70], and provided several instantiations of concrete named families (LegS, LegT, FouT). However, it is not clear whether HIPPO operators can be computed efficiently in either recurrent or convolutional mode, even for these specific cases (except for recurrent LegS, for which a tailored algorithm was given in Lemma 4.6).

6.1.1 A Resolution to Problem 1

While general results from Chapter 5 imply that $\text{hippo}(\omega)$ exists for *any measure* ω (to be precise, any measure with a corresponding family of orthogonal polynomials [197]), resulting in a (LTI) SSM, the efficiency of these is unknown. We resolve this question for recurrent mode.

Theorem 6.1 (Informal). *The memorization operator $\text{hippo}(\omega)$ always has the form $x'(t) = Ax(t) + Bu(t)$ for a low recurrence-width (LRW) [41] state matrix A .*

Although beyond the scope of this chapter, LRW matrices are a type of structured matrix that have linear MVM (Definition 3.1), and whose inverses have linear MVM. Appendix F.1 defines recurrence width and formally restates Theorem 6.1 as Theorem F.4.

By the discussion in Section 3.1.1, Theorem 6.1 resolves Problem 1 for HIPPO-SSMs.

Corollary 6.2. *For all measures that have corresponding orthogonal polynomials, the time-invariant HIPPO operator is an LTI-SSM that is optimally efficient in recurrent mode (Definition 3.1).*

6.1.2 An Attempt for Problem 2

In order to show that HIPPO SSMs can also satisfy Definition 3.2, two things must be done simultaneously: (i) we must find a structure that the \mathbf{A} matrices satisfy, and (ii) we must find a fast algorithm for that structure.

For measures corresponding to the classical orthogonal polynomials (OPs) [197] (in particular, the Jacobi and Laguerre families), there is even more structure than Theorem 6.1.

Corollary 6.3. *For ω corresponding to the classical OPs, $\text{hippo}(\omega)$ is 3-quasiseparable.*

Quasiseparable matrices are a related class of structured matrices with additional algorithmic properties. First of all, it is known that quasiseparable matrices have efficient (linear-time) MVM [160]. We prove that they theoretically also have fast SSM kernels satisfying Definition 3.2, allowing efficient training with convolutions.

Theorem 6.4. *For any k -quasiseparable matrix A (with constant k) and arbitrary B, C , the Krylov function $\mathcal{K}_L(A, B, C)$ can be computed in quasi-linear time and space $\tilde{O}(N + L)$ and logarithmic depth (i.e., is parallelizable). The operation count is in an exact arithmetic model, not accounting for bit complexity or numerical stability.*

Appendix F.1.3 formally define quasiseparable matrices and proves Theorem 6.4. The algorithm for Theorem 6.4 shares similarities to Algorithm 1 (see Remark 6.1) and involves a divide-and-conquer recursion over matrices of *polynomials*, using the observation that (2.8) is related to the power series $\mathbf{C}(\mathbf{I} - \mathbf{A}x)^{-1}\mathbf{B}$.

However, there is a drawback to this algorithm, and thus it is treated more as a *proof of concept* that a HIPPO-structured SSM can be computationally efficient in principle. This is also proven in Appendix F.1.3.

Corollary 6.5. *The algorithm for Theorem 6.4 is over exact arithmetic and is numerically unstable over floating point numbers, where it requires intermediate values exponentially large in the state dimension N .*

Remark 6.1. *We include Theorem 6.4 for several purposes. However, it is not used in*

practice, and it is not necessary to understand this result to understand the main S_4 —HIPPO connections in the remainder of this chapter.

One implication of Theorem 6.4 is that given a particular matrix (e.g. HIPPO), there are many possible structures that it may satisfy that can be leveraged for potential efficient algorithms, and it is not obvious *a priori* what structures will work.

It also underscores the difficulty of Problem 2, illustrating that theoretical efficiency may not be enough and unexpected considerations such as numerical stability may also crop up.

Finally, this result predated the DPLR structure that was later invented to fix the issues with Theorem 6.4, and the algorithm shares several similarities with the DPLR kernel algorithm in Section 3.3, so it is included for historical context.

6.2 DPLR Structure of HIPPO Matrices

In order to address Problem 2 and compute the state space kernel for HIPPO SSMs, it suffices to reduce it to a structured SSM that is known to be efficient. The ideal scenario is if they could be diagonally structured (Section 3.2). By Proposition 3.4, we can hope to apply a state space transformation (i.e. matrix conjugation) to \mathbf{A} and reduce it to diagonal form.

6.2.1 HIPPO Cannot be Diagonalized

Recall that HIPPO specifies a class of certain matrices $\mathbf{A} \in \mathbb{R}^{N \times N}$ that when incorporated into (2.1a), allows the state $x(t)$ to memorize the history of the input $u(t)$. The most important matrix in this class is HIPPO-LegS, which orthogonalizes on an exponential measure (Chapter 5). We redefine this matrix here for convenience.

$$\text{(HIPPO Matrix)} \quad \mathbf{A}_{nk} = - \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k . \\ 0 & \text{if } n < k \end{cases} \quad (6.1)$$

Remark 6.2. When we use the term HIPPO matrix without qualifying the variant (LegS, LegT, FouT), it refers to HIPPO-LegS.

Unfortunately, the naive application of diagonalization on this matrix does not work due

to numerical issues. We derive the explicit diagonalization for the HIPPO matrix (6.1) and show it has entries exponentially large in the state size N , rendering the state space transformation numerically infeasible (e.g. $\mathbf{C}\mathbf{V}$ in Lemma 3.3 would not be computable).

Lemma 6.6. *The HIPPO matrix \mathbf{A} in equation (6.1) is diagonalized by the matrix $\mathbf{V}_{ij} = \binom{i+j}{i-j}$. In particular, $\mathbf{V}_{3i,i} = \binom{4i}{2i} \approx 2^{4i}$. Therefore \mathbf{V} has entries of magnitude up to $2^{4N/3}$.*

6.2.2 Normal Plus Low-Rank (NPLR) Forms of HIPPO

The previous discussion implies that we would ideally only conjugate by well-conditioned matrices \mathbf{V} . The ideal scenario is when the matrix \mathbf{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, Lemma 6.6 shows that it does not contain the HIPPO matrix (6.1) (in fact, that the HIPPO matrix is *far* from normal if measured by condition number of the diagonalization change-of-basis matrix).

Instead, we make the observation that although this HIPPO matrix is not normal, it can be decomposed as the *sum of a normal and low-rank matrix*. This motivates defining the class of **Normal Plus Low-Rank (NPLR)** matrices, which we show in fact includes all named HIPPO matrices that we have defined.

Theorem 6.7. *All HIPPO matrices from Chapters 4 and 5 have an NPLR representation, which is unitarily equivalent to a DPLR representation*

$$\mathbf{A} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^* - \mathbf{P}\mathbf{Q}^\top = \mathbf{V}(\boldsymbol{\Lambda} - (\mathbf{V}^*\mathbf{P})(\mathbf{V}^*\mathbf{Q})^*)\mathbf{V}^* \quad (6.2)$$

for unitary $\mathbf{V} \in \mathbb{C}^{N \times N}$, diagonal $\boldsymbol{\Lambda}$, and low-rank factorization $\mathbf{P}, \mathbf{Q} \in \mathbb{R}^{N \times r}$. These matrices HIPPO-LegS, LegT, FouT all satisfy $r = 1$ or $r = 2$. In particular, equation (6.1) is NPLR with $r = 1$.

6.2.3 DPLR Form

NPLR matrices can be conjugated into diagonal plus low-rank (DPLR) form (now over \mathbb{C} instead of \mathbb{R}). We call the expression $\boldsymbol{\Lambda} - (\mathbf{V}^*\mathbf{P})(\mathbf{V}^*\mathbf{Q})^*$ in equation (6.2) the DPLR form of HIPPO matrices. Theorems 3.5 and 3.6 describe the complexities of SSMs where \mathbf{A} is in DPLR form.

6.2.4 Hurwitz DPLR Form

Finally, in Section 3.3.3 we discussed a small modification of DPLR form that ensures \mathbf{A} is negative semi-definite and thus stable. We show that the HIPPO matrices still satisfy this more restricted structure $\mathbf{A} = \mathbf{\Lambda} - \mathbf{P}\mathbf{P}^*$; in other words, a Hurwitz (stable) S4-DPLR model can still use HIPPO operators.

Corollary 6.8 (Improvement of Theorem 6.7 to Hurwitz form). *All three HIPPO matrices from Chapter 5 are unitarily equivalent to a matrix of the form $\mathbf{A} = \mathbf{\Lambda} - \mathbf{P}\mathbf{P}^*$ for diagonal $\mathbf{\Lambda}$ and $\mathbf{P} \in \mathbb{R}^{N \times r}$ for $r = 1$ or $r = 2$. Furthermore, all entries of $\mathbf{\Lambda}$ have real part 0 (for LegT and FouT) or $-\frac{1}{2}$ (for LegS).*

6.3 Diagonal Approximations of HIPPO

It is remarkable that HIPPO methods can be written in a structured DPLR form, and therefore computed very efficiently when incorporated into an SSM through S4. However, the efficient DPLR computation (Section 3.3) is quite involved and requires a sophisticated algorithm with many linear algebraic techniques that can be difficult to implement.

A natural question is therefore whether there exist effective diagonal SSMs, which can be dramatically simpler to implement and understand. Unfortunately, Chapter 3 discussed how general SSMs (e.g. random diagonal SSMs) are not effective—confirmed later by ablations in Section 7.5—and we showed that HIPPO SSMs cannot be stably transformed in diagonal form (Lemma 6.6). This motivated the necessity of the more complicated DPLR representation.

In this section, we show that there are in fact effective diagonal SSMs. Our main result for diagonal SSMs is the surprising fact that there is a diagonal *approximation* to the main HIPPO(-LegS) matrix that actually recovers the same SSM basis in the limit of infinite state dimension. This instantiation of S4-Diag thus inherits the fully HIPPO-structured S4-DPLR’s mathematical interpretation for modeling long-range dependencies. Strikingly, this approximation is defined by simply chopping off the low-rank term in the DPLR representation.

Inspired by this result, we then propose two even simpler diagonal SSMs with alternate formulas—and in particular, do not require HIPPO to define—that also perform well empirically. These resulting S4D models are extremely simple (e.g. requiring just 2 lines of code to compute the kernel (Section 3.2)), retain the theoretical properties of HIPPO, and

can preserve the performance of the full S4-DPLR + HIPPO model.

6.3.1 Forms of HIPPO-LegS

To set up our main result on the diagonal approximation of LegS, we recap several properties and forms of this matrix. We also recommend revisiting the notation and definitions in Chapter 2, in particular Definition 2.6, Definition 2.7, and Definition 2.18.

Remark 6.3. *As in Section 3.2.1, in the case of diagonal SSMs, \mathbf{A} is diagonal and we will overload notation so that $\mathbf{A}_n, \mathbf{B}_n, \mathbf{C}_n$ directly denote the entries of these parameters. Additionally, for diagonal SSMs, the basis functions in Equation (2.9) satisfy $K_n(t) = e^{t\mathbf{A}_n} \mathbf{B}_n$.*

Recall that the HIPPO-LegS matrix was derived so that the basis kernels $K_n(t)$ have closed-form formulas $L_n(e^{-t})$, where $L_n(t)$ are normalized Legendre polynomials. Consequently, the SSM has a mathematical interpretation of decomposing the input signal $u(t)$ onto a set of infinitely-long basis functions that are orthogonal respect to an exponentially-decaying measure, giving it long-range modeling abilities (Chapter 5).

In Section 6.2 we showed that this \mathbf{A} matrix can be decomposed into the sum of a normal and rank-1 matrix (6.4), which can be unitarily conjugated into a (complex) diagonal plus rank-1 matrix.

$$\begin{aligned} \mathbf{A}_{nk} &= - \begin{cases} (2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} & n > k \\ n+1 & n = k \\ 0 & n < k \end{cases} & \mathbf{A}_{nk}^{(N)} &= - \begin{cases} (n+\frac{1}{2})^{1/2}(k+\frac{1}{2})^{1/2} & n > k \\ \frac{1}{2} & n = k \\ -(n+\frac{1}{2})^{1/2}(k+\frac{1}{2})^{1/2} & n < k \end{cases} \\ \mathbf{B}_n &= (2n+1)^{\frac{1}{2}} & \mathbf{P}_n &= (n+1/2)^{\frac{1}{2}} & \mathbf{A} &= \mathbf{A}^{(N)} - \mathbf{P}\mathbf{P}^\top, & \mathbf{A}^{(D)} &:= \text{eig}(\mathbf{A}^{(N)}) \\ (\text{HIPPO-LegS matrix}) & & & & & (\text{Normal plus low-rank form}) & & (6.4) \end{aligned}$$

Our diagonal approximation of the HIPPO-LegS matrix will be simply *removing the low-rank portion of the DPLR form of the HIPPO-LegS matrix*. More precisely, define the diagonal matrix $\mathbf{A}^{(D)}$ to be the diagonalization of $\mathbf{A}^{(N)}$ in (6.4).

Remark 6.4. *In the development of these models, these various forms of the HIPPO-LegS matrix have been called various names. $\mathbf{A}^{(N)}$ has been called the skew-HIPPO matrix [76], as well as the normal-HIPPO matrix. To be more specific and disambiguate these variants, we may also call $\mathbf{A}^{(N)}$ the HIPPO-LegS-N or HIPPO-N matrix and $\mathbf{A}^{(D)}$ the HIPPO-LegS-D or HIPPO-D matrix.*

Because there are several different concrete matrices with different naming conventions, this table summarizes these special matrices and ways to refer to them.

Matrix	Full Name	Alternate Names
\mathbf{A}	HIPPO-LegS	HIPPO matrix, LegS matrix
$\mathbf{A}^{(N)}$	HIPPO-LegS-N	HIPPO-N, skew-HIPPO, normal-HIPPO
$\mathbf{A}^{(D)}$	HIPPO-LegS-D	HIPPO-D, diagonal-HIPPO

6.3.2 S4D Instantiations

We provide three instantiations of S4D that perform well empirically.

S4D-LegS

The HIPPO-LegS matrix has DPLR representation $\mathbf{A}^{(D)} - \mathbf{P}\mathbf{P}^\top$. Our first result is showing that simply approximating it with $\mathbf{A}^{(D)}$ has a clean mathematical interpretation. Theorem 6.9 shows a surprising fact that does not hold in general for DPLR matrices (Appendix F.3.1), and arises out of the special structure of this particular matrix.

Theorem 6.9. *Let $\mathbf{A} = \mathbf{A}^{(N)} - \mathbf{P}\mathbf{P}^\top$ and \mathbf{B} be the HIPPO-LegS matrices, and $K_{\mathbf{A}, \mathbf{B}}(t)$ be its basis. As the state size $N \rightarrow \infty$, the SSM basis $K_{\mathbf{A}^{(N)}, \mathbf{B}/2}(t)$ limits to $K_{\mathbf{A}, \mathbf{B}}(t)$ (Figure 6.1).*

Note that $\mathbf{A}^{(N)}$ is then *unitarily* equivalent to $\mathbf{A}^{(D)}$, which preserves the stability and timescale (Section 5.4) of the system.

We define **S4D-LegS** to be the S4D method for this choice of diagonal $\mathbf{A} = \mathbf{A}^{(D)}$. Empirically, general results in Chapter 7 show that this system performs quite close to S4(-DPLR)-LegS, but is often slightly worse. This phenomenon makes intuitive sense from this theoretical result, since by Theorem 6.9 S4D-LegS is a noisy approximation to S4-LegS. Figure 6.1 illustrates this result, and also shows a curious phenomenon involving different discretization rules that is open for future work.

S4D-Inv

To further simplify S4D-LegS, we analyze the structure of $\mathbf{A}^{(D)} = \text{diag}\langle \mathbf{A} \rangle$ in more detail. The real part is easy to understand, which follows from the proof of Theorem 6.7:

Lemma 6.10. $\Re(\mathbf{A}) = -\frac{1}{2}\mathbf{1}$

Let the imaginary part be sorted, i.e. $\Im(\mathbf{A})_n$ is the n -th largest (positive) imaginary component. We empirically deduced the following conjecture for the asymptotics of the imaginary part.

Conjecture 6.11. *As $N \rightarrow \infty$, $\Im(\mathbf{A})_0 \rightarrow \frac{1}{\pi}N^2 + c$ where $c \approx 0.5236$ is a constant. For a fixed N , the other eigenvalues satisfy an inverse scaling in n : $\Im(\mathbf{A})_n = \Theta(n^{-1})$.*

Figure 6.2 empirically supports this conjecture. Based on Conjecture 6.11, we propose the initialization S4D-Inv to use the following inverse-law diagonal matrix which closely approximates S4D-LegS.

$$(\text{S4D-Inv}) \quad \mathbf{A}_n = -\frac{1}{2} + i\frac{N}{\pi} \left(\frac{N}{2n+1} - 1 \right) \quad (6.5)$$

S4D-Lin

While S4D-Inv can be seen as an approximation to the original S4-LegS, we propose an even simpler scaling law for the imaginary parts that can be seen as an approximation of S4-FouT ([75]), where the imaginary parts are simply the Fourier series frequencies (i.e. matches the diagonal part of the DPLR form of S4-FouT). Figure 3.1 (Right) illustrates the S4D-Lin basis $e^{t\mathbf{A}}\mathbf{B}$, which are simply damped Fourier basis functions.

$$(\text{S4D-Lin}) \quad \mathbf{A}_n = -\frac{1}{2} + i\pi n \quad (6.6)$$

General Diagonal SSM Basis Functions

The empirical study in Section 7.5.4 performs many ablations of different diagonal initializations, including some additional diagonal initializations plotted in Figure 6.2, showing that many natural variants of the proposed methods do not perform as well. The overall guiding principles for the diagonal state matrix \mathbf{A} are twofold, which can be seen from the closed form of the basis functions $K_n(t) = e^{t\mathbf{A}_n}\mathbf{B}_n$.

First, the real part of \mathbf{A}_n controls the decay rate of the function. $\mathbf{A}_n = -\frac{1}{2}$ is a good default that bounds the basis functions by the envelope $e^{-\frac{t}{2}}$, giving a constant timescale (Figure 3.1 (Right)).

Second, the imaginary part of \mathbf{A}_n controls the oscillating frequencies of the basis function. Critically, these should be spread out, which explains why random initializations of \mathbf{A}

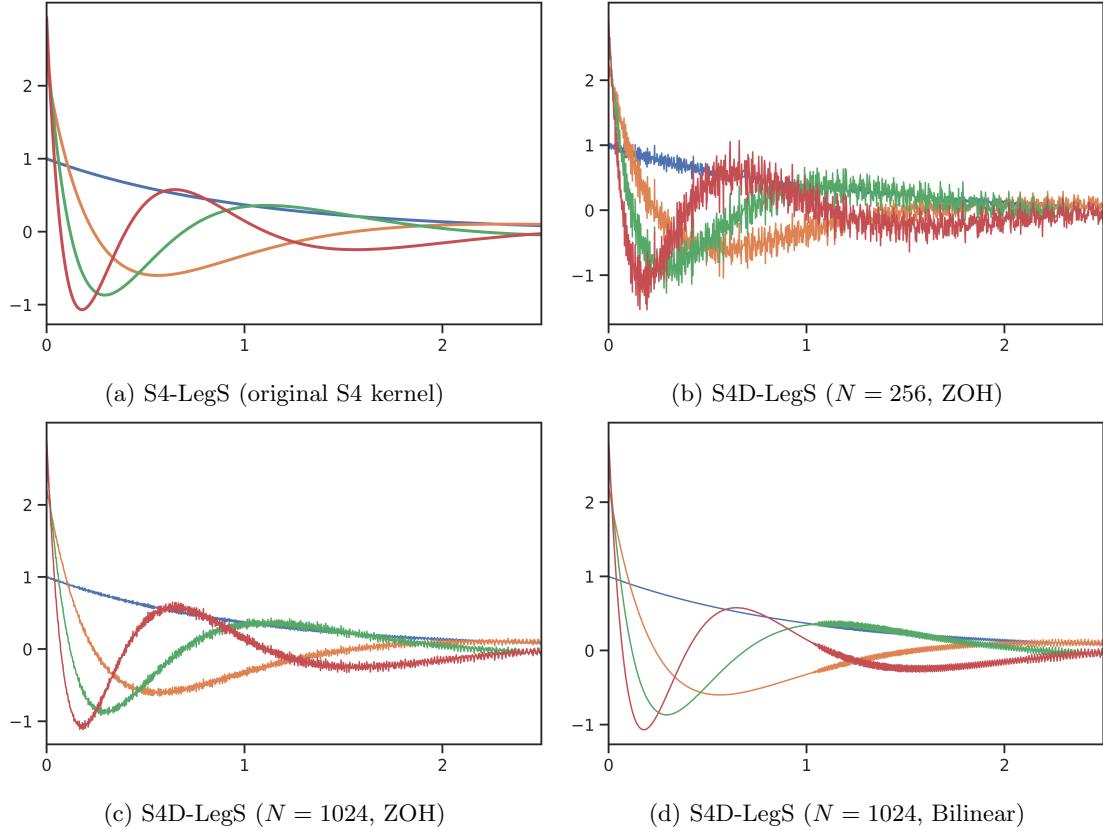


Figure 6.1: (Visualization of Theorem 6.9.) (a) The particular (\mathbf{A}, \mathbf{B}) matrix chosen in S4 results in smooth basis functions $e^{t\mathbf{A}}\mathbf{B}$ with a closed form formula in terms of Legendre polynomials. By the HIPPO theory, convolving against these functions has a mathematical interpretation as orthogonalizing against an exponentially-decaying measure. (b, c) By special properties of this state matrix, removing the low-rank term of its NPLR representation produces the same basis functions as $N \rightarrow \infty$, explaining the empirical effectiveness of DSS. (c) Curiously, the bilinear transform instead of ZOH smooths out the kernel to exactly match S4-LegS as N grows.

do not perform well. S4D-Inv and S4D-Lin use simple asymptotics for these imaginary components that provide interpretable bases. We believe that alternative initializations that have different mathematical interpretations may exist, which is an interesting question for future work.

6.4 Discussion: HIPPO vs. S4

HIPPO and S4 have distinct meanings and can be motivated completely independently, yet are also closely related and their connections can be confusing. This section recapitulates

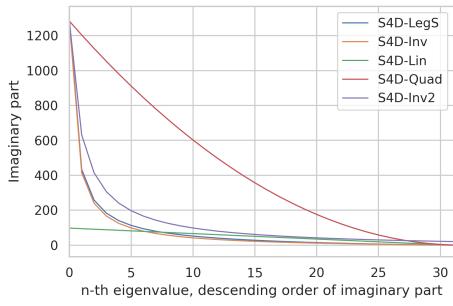


Figure 6.2: (**Eigenvalues of S4D variants.**) All S4D methods have eigenvalues $-\frac{1}{2} + \lambda_n i$. S4D-LegS theoretically approximates dynamics of the original (non-diagonal) S4 (Blue), and has eigenvalues following an inverse law $\lambda_n \propto n^{-1}$ (Orange). The precise law is important: other scaling laws with the same range, including an inverse law with different constant (Purple) and a quadratic law (Red), perform empirically worse (Section 7.5.4). A very different linear law based on Fourier frequencies also performs well (Green).

their meaning and discusses various facets of their relationship.

6.4.1 Instantiations of HIPPO

HIPPO is a framework for online memorization (Chapter 4), and an orthogonal SSM (O-SSM) is a formalism of it in the language of SSMs. We use these terms synonymously.

Every specific instance of HIPPO is denoted HIPPO-[Name], which refers to a pair $(\mathbf{A}(t), \mathbf{B}(t))$ that is an orthogonal SSM (Definition 5.1). This can also be shortened to just [Name] (e.g. LegT, LegS, FouT), which usually abbreviate the corresponding basis functions (e.g. scaled Legendre, truncated Legendre).

For LTI SSMs and LSI SSMs (Definition 4.4), since the dependence on t is non-existent or straightforward, we can also drop it and have HIPPO-[Name] or [Name] refer to a static pair of (\mathbf{A}, \mathbf{B}) matrices, as in equations (5.1), (5.2), (5.3).

HIPPO-LegS Ambiguity

We remark that LegS is a bit of a special case.

One of the main results of Chapter 5 is that the LegS matrices (5.1) can be incorporated in multiple ways into HIPPO operators. In particular, $(\frac{1}{t}\mathbf{A}, \frac{1}{t}\mathbf{B})$ is an LSI orthogonal SSM (Section 4.3) with a scaled uniform measure, and (\mathbf{A}, \mathbf{B}) is an LTI orthogonal SSM (or TO-SSM) (Section 5.2) with an exponentially decaying measure.

We usually overload notation so that LegS refers simply to the pair of matrices in (5.1). However, “HIPPO-LegS” is actually ambiguous in general and its usage in an O-SSM should be disambiguated. This is usually not an issue because “S4-LegS” is not ambiguous, discussed in Section 6.4.3.

6.4.2 Instantiations of S4

S4 broadly refers to a structured SS(S)M layer. Unlike HIPPO, which defines a static SSM (\mathbf{A}, \mathbf{B}) (see Definition 2.6), S4 defines an SSM $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ which is meant to be incorporated as a sequence model and trained. S4 models are disambiguated by the structure on \mathbf{A} (and possibly \mathbf{B}), which are also often instantiated (or initialized) in particular ways. Thus S4-[Name] means use the SSM [Name] for (\mathbf{A}, \mathbf{B}) , and add trainable parameters \mathbf{C} and Δ .

Note that these instantiations do not have to be HIPPO operators. In particular, the S4D-Inv and S4D-Lin methods defined in Section 6.3 are not HIPPO methods *per se* in the technical sense of orthogonal SSM, although they are related to and motivated from HIPPO.

6.4.3 Combining S4 and HIPPO

The main results of this chapter are about when orthogonal SSMs (HIPPO) are also structured SSMs (S4). This remarkable intersection simultaneously endows SSMs with principled and interpretable state representations, and is also computationally efficient, resolving two main challenges with SSMs.

S4-LegS

We highlight the particular case of S4-LegS. Although we mentioned that HIPPO-LegS is ambiguous, since S4 is defined as an LTI SSM (Remark 2.6) for its convolutional view to be valid, the method S4-LegS unambiguously refers to the SSM (\mathbf{A}, \mathbf{B}) .

This is also the main and original form of S4, and generally is a safe default for empirical performance compared to other variants. Thus the unqualified name S4 often refers to this variant.

Frozen vs. Trainable Matrices

One detail that has not been fully discussed is resolving the discrepancy between HIPPO being a static SSM (\mathbf{A}, \mathbf{B}) , and S4 being a trainable SSM $(\mathbf{A}, \mathbf{B}, \mathbf{C})$. Note that because orthogonal and structured (e.g. HIPPO and DPLR matrices) do not necessarily coincide, training any S4 model—even if initialized to a HIPPO formula—no longer guarantees that it remains an orthogonal SSM after (and during) training.

Table 6.1: Summary of named S4 variants.

S4 Variant	Structure	Initialization of (\mathbf{A}, \mathbf{B})
S4D-LegS	Diag	$\mathbf{A}^{(D)}$ in (6.4)
S4D-Inv	Diag	equation (6.5)
S4D-Lin	Diag	equation (6.6)
S4D-Rand	Diag	Random
S4-LegS	DPLR	equation (5.1)
S4-LegT	DPLR	equation (5.2)
S4-FouT	DPLR	equation (5.3)
S4-Rand	DPLR	Random

We take the following perspective on this. *If* an SSM was not allowed to train its (\mathbf{A}, \mathbf{B}) matrices, then setting them to a HIPPO method is much more principled and effective. Freezing these matrices should also be fine by the perspective in Definition 2.18: they simply define a set of nice basis functions, and learning \mathbf{C} is enough to represent any convolution (in the limit of state size N). However, training parameters generally results in better performance in deep learning, and so it does not hurt; one can perhaps view it as a form of fine-tuning.

When talking about defining S4 models, we often use the term *instance* to keep the meaning flexible. Thus instantiating HIPPO means a fixed (\mathbf{A}, \mathbf{B}) method, while instantiating S4 can also mean fixing (\mathbf{A}, \mathbf{B}) , or initializing them in a special way and continuing to train them.

Section 7.5 includes several ablations on the trainability of SSM parameters.

6.4.4 Summary of Named S4 Variants

Table 6.1 summarizes the named S4 variants defined in this thesis. As per the conventions discussed in this section, the possible instantiations are not restricted to this; we include “random” instantiations of both S4-DPLR and S4-Diag as an example, which are used in ablations in Chapter 7.

Part III

Applications and Extensions

Chapter 7

Empirical Evaluation: S4 as a General Sequence Model

This chapter evaluates the empirical performance of S4 on a variety of sequence modeling tasks across many domains (e.g. images, audio, text, time-series) and capabilities (e.g. large-scale training, fast generative modeling, atypical sampling, forecasting).

- Section 7.1 summarizes the main training details of a core S4 layer.
- Section 7.2 describes complete details of a simple SSNN, or deep neural network architecture built around S4 layers. This is used throughout our experiments.
- Section 7.3 contains the main experimental results, focusing on the empirical performance of S4 across a wide variety of standard benchmarks for sequence models in the literature.
- Section 7.4 highlights capabilities of S4 derived from its theoretical properties: its continuous, recurrent, and convolutional representations, as well as principled memory representations.
- Section 7.5 performs comprehensive ablations of several details of S4’s definition, validating effects predicted by our theory.

7.1 Summary of S4 Details

For completeness, we summarize all details of parameterizing a linear S4 layer, with references to the chapters that discuss them.

Parameterization

We consider Diagonal and DPLR structured state space models (S4-Diag and S4-DPLR, or S4D and S4), with algorithms described in Section 3.2 and Section 3.3 respectively. Other details of parameterizing the models are discussed in Section 3.4.

Concretely, an S4-Diag layer's parameters are $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{C}^{N/2}$ (overloading notation since they have redundant dimensions; see Remark 3.6), and $\Delta \in \mathbb{R}$, for a total of $3N+1$ trainable (real) parameters per layer per input channel. An S4-DPLR layer has parameters $\mathbf{\Lambda}$ and \mathbf{P} representing $\mathbf{A} = \mathbf{\Lambda} - \mathbf{P}\mathbf{P}^*$, for $4N + 1$ parameters. (We henceforth ignore the $+1$ when describing parameter counts.)

Initialization of A and B

All S4 variants are initialized to specific formulas for (\mathbf{A}, \mathbf{B}) (Chapters 4 to 6). All variants considered in this thesis and our experiments are defined in Table 6.1.

Initialization of C

\mathbf{C} is initialized randomly (e.g. with standard Normal distribution) with standard deviation 1, which is variance-preserving for S4's (\mathbf{A}, \mathbf{B}) initializations (Section 5.4). This is in contrast to standard deep learning initializations, which scale with the dimension (such as Xavier initialization $N^{-\frac{1}{2}}$ [66]).

Initialization of Δ

Δ is initialized geometric-randomly to capture a desired length of dependencies (Section 5.4). The range is usually $(\Delta_{min}, \Delta_{max}) = (0.001, 0.1)$ by default for tasks with long sequences (1000+). Note that because of broadcasting over the H input channels or heads, each head is initialized to a different Δ and many timescales are covered.

7.2 A Simple SSNN Architecture

The standard S4 model is a SISO SSM, or a linear map on 1-D sequences $\mathbb{R}^L \rightarrow \mathbb{R}^L$. It can be flexibly incorporated into a deep neural network architecture. We describe the simplest way to extend S4 into an SSNN similar to standard deep sequence models such as ResNets [80] and Transformers [217]. The experiments in the remainder of this section all use this architecture.

7.2.1 Multiple Channels

Typically, DNNs operate on inputs with multiple features or channels (Figure 1.1). As described in Section 2.3.5, S4 handles multiple channels by simply processing each one with an independent model.

Note that the parameters are simply broadcasted with an extra H dimension, for a total of $4HN$ parameters for the S4 layer (or $3HN$ for S4D).

7.2.2 Parameter Tying

We note that parameters can be shared between multiple channels to reduce parameters. We sometimes tie the (\mathbf{A}, \mathbf{B}) parameters across the H channels, which has several motivations.

First, the interpretation in Definition 2.18 suggests that it is sufficient to define a single convolutional basis (\mathbf{A}, \mathbf{B}) ; as long as each of the \mathbf{C} parameters is independent between the H channels, each channel can learn a different linear combination of the basis functions to define independent convolution kernels.

Second, Part II develops several (\mathbf{A}, \mathbf{B}) instantiations with theoretical interpretations, and suggest that a single fixed (\mathbf{A}, \mathbf{B}) parameter (which doesn't necessarily even need to be trained) should be enough.

Finally, tying the \mathbf{A} matrix reduces the difference between the DPLR and Diagonal parameterizations to just a negligible N parameters per layer, and the difference between trainable and frozen variants (Section 6.4.3) to just $3N$ or $4N$ parameters per layer. This allows for controlled comparisons between parameterizations in our ablations.

7.2.3 Linear Mixing

Since different input channels do not interact at all, a (square) linear feed-forward layer is applied after the S4 layer to mix the H channels. The overall parameter count is therefore $O(H^2) + O(HN)$ parameters per layer.

We also often apply a GLU activation afterwards. Instead of a single linear layer $\mathbf{W}y$, the GLU version is defined as $(\mathbf{W}_1y) \circ \sigma(\mathbf{W}_2y)$ for $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{H \times H}$ [38]. These have been empirically found to improve linear layers of deep neural networks in general [186]. Note that this also doubles the parameters of this linear layer, from H^2 to $2H^2$.

7.2.4 Nonlinear Activations

As the rest of the model is linear, a nonlinearity is inserted between the S4 and linear mixing layer. We always use the GELU activation function [82].

7.2.5 Residuals and Normalization

Multiple S4 layers are stacked using standard architectures for deep neural networks, similar to ResNets [81] and Transformers [217]. In particular, we apply a normalization layer per block (either BatchNorm or LayerNorm [92, 7]) and stack blocks with residual connections. The normalization placement can be either pre-norm or post-norm [168], which is treated as a hyperparameter that potentially affects performance and stability [127, 39].

7.2.6 Bidirectional Modeling

Like ODEs and RNNs, S4 is a causal or *unidirectional* model (Definition 2.5), which is necessary for some settings such as autoregressive modeling (Section 2.1.3). However, many settings may not require causality (such as basic classification tasks). We provide a simple *bidirectional* relaxation to better take advantage of available global context in non-causal settings.

One standard approach is to pass the input sequence through an S4 layer, and also reverse it and pass it through an independent second layer. These outputs are concatenated and passed through a positionwise linear layer as in the standard S4 block.

$$y = \text{Linear}(\text{Concat}(\text{S4}(x), \text{reverse}(\text{S4}(\text{reverse}(x)))))$$

We also propose another approach that can be more computationally efficient, which combines both convolution kernels back-to-back into a single one, so only one FFT-convolution call is needed instead of two.

This can be viewed as an approximation of the reverse-and-concatenate approach as follows. The above output can be written as

$$y = \text{Linear}(\text{S4}(x)) + \text{Linear}(\text{reverse}(\text{S4}(\text{reverse}(x)))).$$

By tying the Linear parameters, this can be approximated by

$$y = \text{Linear}(\text{S4}(x) + \text{reverse}(\text{S4}(\text{reverse}(x)))).$$

Finally, since $\text{S4}(x) = K * x$ is a convolution for some kernel K , this is equivalent to

$$\begin{aligned} y &= \text{Linear}(K_1 * x + \text{reverse}(K_2 * \text{reverse}(x))) \\ &= \text{Linear}(K_1 * x + \text{reverse}(K_2) * x) \\ &= \text{Linear}((K_1 + \text{reverse}(K_2)) * x) \end{aligned}$$

for some kernels K_1 and K_2 . This reduces the number of convolutions from two to one, which noticeably improves the computational efficiency of the full model.

7.3 Empirical Results

This section shows S4’s strong empirical performance in a wide variety of settings and tasks. We highlight that many of these benchmarks are typically targeted by specialized models, while we use the same generic SSNN, validating S4’s performance as a general sequence model that requires weaker inductive bias.

Sections 7.3.1 to 7.3.3 present benchmarks on classification and regression tasks on image, time-series, and speech modalities of lengths 1000 to 16000. Section 7.3.4 highlights S4’s performance on the Long Range Arena benchmark for long-range reasoning. Sections 7.3.5 and 7.3.6 show that S4 can be applied with minimal modifications on more targeted capabilities including time-series forecasting and generative modeling.

7.3.1 Sequential Image Classification

We evaluate S4 on the popular sequential MNIST (sMNIST), permuted MNIST (pMNIST), and sequential CIFAR-10 (sCIFAR) benchmarks (Table 7.1). The original intention of these tasks is to sequentially process an image pixel-by-pixel before using the final state to generate a classification label, thus testing the ability of recurrent models to remember long-term dependencies of length up to 1k [5]. Equivalently, images are flattened in row-major order into a single sequence before being processed by a generic 1-D sequence model. The permuted variant (pMNIST) applies a fixed permutation to every input, further increasing the difficulty by breaking locality.

Beyond long-range dependencies, these benchmarks point to a recent effort of the ML community to solve vision problems with reduced domain knowledge, in the spirit of models such as Vision Transformers [51] and MLP-Mixer [205] which involve patch-based models with less 2-D inductive bias. However, this goal can be difficult: for example, sCIFAR is a particularly challenging dataset where all prior sequence models have a gap of over 15% to a simple 2-D CNN.

Table 7.1 compares the test accuracy of S4 against reported results from the literature, including many RNN-, CNN-, and Transformer- based models. On sCIFAR, S4 sets state of the art by almost 20% and is competitive with a comparable sized ResNet18—a 2-D CNN—which attains 89.46% (without data augmentation, to be comparable to the standard sCIFAR setting).

7.3.2 Time Series Regression

We use the BIDMC healthcare datasets (Table 7.2), a suite of time series regression problems of length 4000 on estimating vital signs, which are of major importance to modern healthcare applications. Inputs are time series of length 4000 with two channels (EKG and PPG signals), which are used to predict three regression targets: heart rate (HR), respiratory rate (RR), and blood oxygen saturation (SpO2).

S4 substantially improves on all baselines, including both deep learning baselines (RNN, CNN, and Transformer) as well as other popular machine learning methods for time series, reducing RMSE by 76% to 92% on the three targets compared to all baselines.

Table 7.1: (**Pixel-level 1-D image classification.**) Comparison against reported test accuracies from prior works (Transformer, CNN, RNN, and SSM models). Citations refer to the original model; additional citation indicates work from which this baseline is reported.

Model	sMNIST	pMNIST	sCIFAR
Transformer [217, 213]	98.9	97.9	62.2
CKConv [174]	99.32	<u>98.54</u>	63.74
TrellisNet [12]	99.20	98.13	73.42
TCN [11]	99.0	97.2	-
LSTM [86, 72]	98.9	95.11	63.01
r-LSTM [213]	98.4	95.2	72.2
Dilated GRU [22]	99.0	94.6	-
Dilated RNN [22]	98.0	96.1	-
IndRNN [123]	99.0	96.0	-
exprNN [121]	98.7	96.6	-
UR-LSTM	99.28	96.96	71.00
UR-GRU [72]	99.27	96.51	<u>74.4</u>
LMU [218]	-	97.15	-
HIPPO-RNN [70]	98.9	98.3	61.1
UNIcoRNN [178]	-	98.4	-
LMUFFT [28]	-	98.49	-
LipschitzRNN [56]	99.4	96.3	64.2
S4	99.63	98.70	91.13

7.3.3 Speech Classification

Speech is a typical real-world time series domain, involving signals sampled from an underlying physical process at high frequencies, resulting in very long sequences that are challenging for ML models.

Table 7.3 shows speech classification for two settings on the *Speech Commands* dataset.

Dataset Details

Speech Commands (SC) [222] is a keyword spotting (i.e. speech recognition) dataset with the goal of classifying audio recordings into one of 35 spoken English words such as ‘left’, ‘right’, etc. The data consists of 1-second audio clips sampled at 16000 Hz, so inputs are sequences of length 16000 with 1 channel. A subsampled variant called SC10 using a 10-class subset has been commonly used in the literature to provide a simpler and more class-balanced version of the dataset [107, 174, 74, 173].

Table 7.2: (**BIDMC Vital signs prediction.**) RMSE (std.) for predicting respiratory rate (RR), heart rate (HR), and blood oxygen (SpO2). (*Top*) S4. (*Middle*) Deep learning baselines. (*Bottom*) Other popular machine learning methods for time series. Citations indicate reported numbers from prior work.

Model	HR	RR	SpO2
S4	0.332 (0.013)	0.247 (0.062)	<u>0.090</u> (0.006)
S4-FouT	<u>0.339</u> (0.020)	0.301 (0.030)	0.068 (0.003)
S4D-LegS	0.367 (0.001)	0.248 (0.036)	0.102 (0.001)
S4D-Inv	0.373 (0.024)	0.254 (0.022)	0.110 (0.001)
S4D-Lin	0.379 (0.006)	<u>0.226</u> (0.008)	0.114 (0.003)
UnICORNN [178]	1.39	1.06	0.869
coRNN [178]	1.81	1.45	-
CKConv	2.05	1.214	1.051
NRDE [140]	2.97	1.49	1.29
LSTM [178]	10.7	2.28	-
Transformer	12.2	2.61	3.02
XGBoost [199]	4.72	1.67	1.52
Random Forest [199]	5.69	1.85	1.74
Ridge Regress. [199]	17.3	3.86	4.16

SC10 Results

Traditional systems involve complex pipelines that require feeding hand-crafted features into DNNs [172]. Table 7.3a reports results for the SC10 subset [107, 174] for 10-way classification of 1-second audio clips. Most sequence models are unable to use the raw speech signal effectively, instead requiring pre-processing with standard mel-frequency cepstrum coefficients (MFCC) to short sequences (length 161). By contrast, S4 sets SOTA on this dataset while training on the raw signal (length 16000). We note that MFCC extracts sliding window frequency coefficients and thus is related to the coefficients $x(t)$ defined by HIPPO. Consequently, S4 may be interpreted as automatically learning MFCC-like features in a trainable basis.

SC (Full) Results

We also use the full 35-class Speech Commands dataset, which we advocate for follow-up work because it is more standard and avoids dataset fragmentation in the literature. As we are not aware of a collection of strong baselines for raw waveform classification using the full dataset (there do exist baselines trained on MFCC), we trained several baselines from scratch for Table 7.3b. The InceptionNet, ResNet-18, and XResNet-50 models are 1D adaptations from Nonaka and Seita [144] of popular CNN architectures originally for

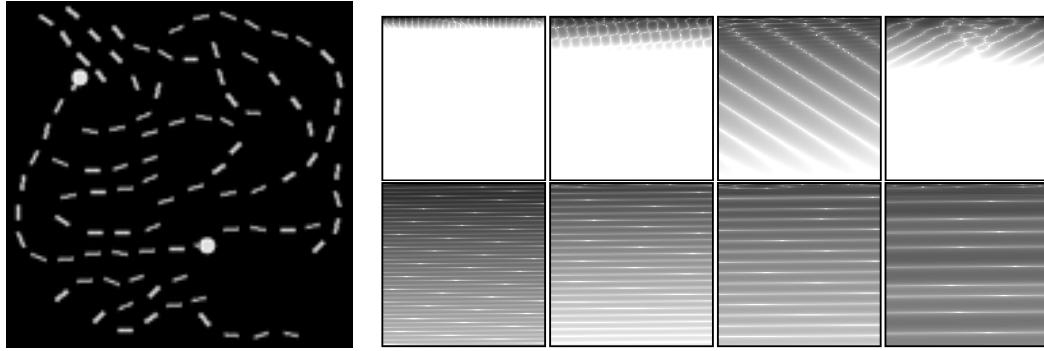


Figure 7.1: Visualizations of a trained S4 model on LRA Path-X. SSM convolution kernels $\bar{K} \in \mathbb{R}^{16384}$ are reshaped into a 128×128 image. (Left) Example from the Path-X task, which involves deducing if the markers are connected by a path (Top) Filters from the first layer (Bottom) Filters from the last layer.

vision. The ConvNet architecture is a very large custom CNN using modern best practices for deep neural networks that we tuned for strong performance; architecture details are in Appendix G.1.3.

Table 7.3: (**Speech Commands classification.**) Test accuracy on 10- or 35- way keyword spotting. (MFCC) Training on MFCC-processed features (length 161). (16k) Training examples are 1-second audio waveforms sampled at 16000Hz, i.e. a 1 channel sequence of length 16000. (8k) Sampling rate change: 0-shot testing at 8000Hz where examples are constructed by naive decimation. \times denotes not applicable or computationally infeasible on single GPU.

(a) (**SC10 subset.**) Transformer, NODE, RNN, CNN, and SSM models.

	MFCC	16k Hz	8k Hz
Transformer	90.75	\times	\times
Performer	80.85	30.77	30.68
ODE-RNN	65.9	\times	\times
NRDE	89.8	16.49	15.12
UniCORNN	90.64	11.02	11.07
ExpRNN	82.13	11.6	10.8
CKConv	95.3	71.66	65.96
WaveGAN-D	\times	<u>96.25</u>	\times
S4	<u>93.96</u>	98.32	96.30

(b) (**Full dataset.**) S4 (DPLR and Diag) variants, and a collection of CNN baselines.

Model	Param.	16000Hz	8000Hz
S4-LegS	307K	96.08 (0.15)	91.32 (0.17)
S4-FouT	307K	95.27 (0.20)	91.59 (0.23)
S4D-LegS	306K	95.83 (0.14)	91.08 (0.16)
S4D-Inv	306K	<u>96.18</u> (0.27)	91.80 (0.24)
S4D-Lin	306K	96.25 (0.03)	<u>91.58</u> (0.33)
InceptionNet	481K	61.24 (0.69)	05.18 (0.07)
ResNet-18	216K	77.86 (0.24)	08.74 (0.57)
XResNet-50	904K	83.01 (0.48)	07.72 (0.39)
ConvNet	26.2M	95.51 (0.18)	07.26 (0.79)

Table 7.4: (**Long Range Arena.**) Test accuracy on full suite of LRA tasks. Hyperparameters in Appendix G.1.2 \times denotes failure to learn better than random guessing, following convention from Tay et al. [202]. (*Top*) Original Transformer variants in LRA [202]. (*Middle*) Other models reported in the literature. (*Bottom*) S4 variants.

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	\times	53.66
Local Attention	15.82	52.98	53.39	41.46	66.63	\times	46.71
Sparse Trans.	17.07	63.58	59.59	44.24	71.71	\times	51.03
Longformer	35.63	62.85	56.89	42.22	69.71	\times	52.88
Linformer	35.70	53.94	52.27	38.56	76.34	\times	51.14
Reformer	<u>37.27</u>	56.10	53.40	38.07	68.50	\times	50.56
Sinkhorn Trans.	33.67	61.20	53.83	41.23	67.45	\times	51.23
Synthesizer	36.99	61.68	54.67	41.61	69.45	\times	52.40
BigBird	36.05	64.02	59.29	40.83	74.87	\times	54.17
Linear Trans.	16.13	<u>65.90</u>	53.09	42.34	75.30	\times	50.46
Performer	18.01	65.40	53.82	42.77	77.05	\times	51.18
FNet	35.33	65.11	59.61	38.67	<u>77.80</u>	\times	54.42
Nyströmformer	37.15	65.52	<u>79.56</u>	41.58	70.94	\times	57.46
Luna-256	37.25	64.57	79.29	<u>47.38</u>	77.72	\times	<u>59.37</u>
S4D-Rand	60.08	86.93	90.39	83.80	91.12	95.70	84.67
S4D-Lin	60.76	87.36	90.66	<u>89.21</u>	95.52	97.26	<u>86.80</u>
S4D-Inv	60.36	87.47	90.62	88.57	94.98	97.37	86.56
S4D-LegS	<u>60.99</u>	87.34	90.65	89.04	95.35	<u>97.33</u>	86.78
S4-Rand	60.46	<u>87.94</u>	<u>90.87</u>	84.59	91.57	95.59	85.17
S4-FouT	61.21	86.72	90.47	89.18	<u>95.68</u>	\times	78.88
S4(-LegS)	60.86	88.73	91.14	89.49	95.76	97.32	87.22

7.3.4 Long Range Arena (LRA)

Long Range Arena (LRA)

LRA [202] contains 6 tasks with lengths 1K-16K steps, encompassing modalities and objectives that require similarity, structural, and visuospatial reasoning. Table 7.4 compares S4 against the 11 Transformer variants from Tay et al. [202] as well as follow-up work. S4 substantially advances the SOTA, outperforming all baselines on all tasks and averaging 87% compared to less than 60% for every baseline. Notably, S4 solves the Path-X task, an extremely challenging task that involves reasoning about long range dependencies over sequences of length $128 \times 128 = 16384$. All previous models have failed (i.e. random guessing) due to memory or computation bottlenecks, or simply being unable to learn such long dependencies.

Table 7.4 also shows that S4-Diag variants are highly competitive with S4-DPLR on all

datasets. We also include baselines using the S4(D) parameterizations but with random initializations to show the difference that our initializations can make, particularly on the image-based tasks.

Remark 7.1. *S4-FouT is the only S4 variant that does not learn on Path-X, which we hypothesize is because it is a finite window method (Section 5.3). Section 7.5.7 ablates HIPPO methods and shows tasks that S4-FouT is strong at.*

Path-X Visualizations

We analyze S4’s performance on Path-X by visualizing its learned representations, in particular 1-D convolution kernels $\bar{\mathbf{K}}$. Figure 7.1 shows that S4 learns a variety of filters that display spatially consistent structure and demonstrate awareness of the 2-D nature of the data. In particular, the lower layers learn simple kernels that extract features from just a few rows of local context while ignoring the rest of the image. On the other hand, higher layers aggregate information globally across full columns of the image at varying spatial frequencies. Filters in these higher layers span the entire context (16384 pixels), supporting S4’s ability to learn LRDs.

7.3.5 Time Series Forecasting

Time series forecasting is a domain that typically requires specialized domain-specific pre-processing and architectures. We compare S4 to the Informer [241], a new Transformer architecture that uses a complex encoder-decoder designed for time series forecasting problems. A simple application of S4 that treats forecasting as a masked sequence-to-sequence transformation outperforms the Informer and other baselines on 40/50 settings across 5 forecasting tasks. Notably, S4 is better on the longest setting in each task, e.g. reducing MSE by 37% when forecasting 30 days of weather data (Table 7.5).

Table 7.5: Univariate long sequence time-series forecasting results. Full results in Appendix G.1.4.

S4		Informer		LogTrans		Reformer		LSTM		DeepAR		ARIMA		Prophet	
		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

Figure 7.2 contrasts the simple architecture of S4 against that of the Informer [241].

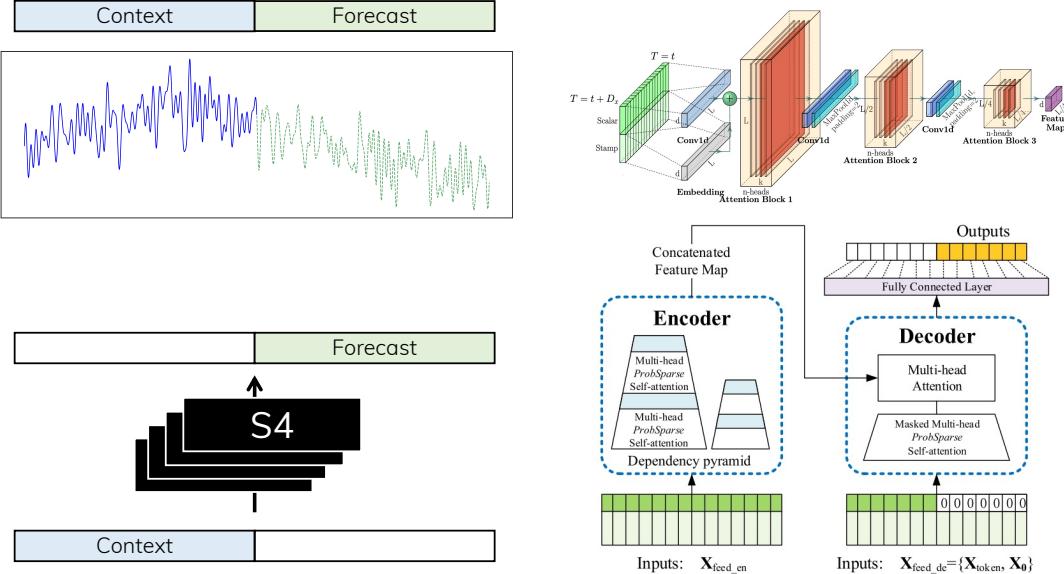


Figure 7.2: Comparison of S4 and specialized time-series models for forecasting tasks. (*Top Left*) The forecasting task involves predicting future values of a time-series given past context. (*Bottom Left*) We perform simple forecasting using a sequence model such as S4 as a black box. (*Right*) Informer uses an encoder-decoder architecture designed specifically for forecasting problems involving a customized attention module (figure taken from Zhou et al. [241]).

In Figure 7.2, the goal is to forecast a contiguous range of future predictions (*Green*, length F) given a range of past context (*Blue*, length C). We simply concatenate the entire context with a sequence of masks set to the length of the forecast window. This input is a single sequence of length $C+F$ that is run through the same simple deep S4 model used throughout this chapter, which maps to an output of the same dimensions (Definition 2.1), i.e. length $C+F$. We then grab just the last F outputs as the forecasted predictions.

7.3.6 Large-scale Autoregressive Generative Modeling

We investigate two well-studied image and text benchmarks to validate the scalability, flexibility, and efficiency of S4. These tasks require much larger models than our previous tasks – up to 250M parameters.

Table 7.6: (**CIFAR-10 density estimation.**) As a generic sequence model, S4 is competitive with previous autoregressive models in bits per dim. (bpd) while incorporating no 2D inductive bias, and has fast generation through its recurrence mode. Baseline results are directly reported numbers from prior work.

Model	bpd	2D bias	Images / sec. (throughput)
Transformer [104]	3.47	None	0.32 (1×)
Linear Transformer [104]	3.40	None	17.85 (56×)
PixelCNN [215]	3.14	2D conv.	-
Row PixelRNN [216]	3.00	2D BiLSTM	-
PixelCNN++ [182, 170]	2.92	2D conv.	<u>19.19</u> (59.97×)
Image Transformer [155]	2.90	2D local attn.	0.54 (1.7×)
PixelSNAIL	<u>2.85</u>	2D conv. + attn.	0.13 (0.4×)
Sparse Transformer [27]	2.80	2D sparse attn.	-
S4 (base)	2.92	None	20.84 (65.1×)
S4 (large)	<u>2.85</u>	None	3.36 (10.5×)

CIFAR-10 Density Estimation

First, CIFAR density estimation is a popular benchmark for autoregressive models, where images are flattened into a sequence of 3072 RGB subpixels that are predicted one by one. Table 7.6 shows that *with no 2D inductive bias*, S4 is competitive with the best models designed for this task.

WikiText-103 Language Modeling

Second, WikiText-103 is an established benchmark for language modeling, an important task for large-scale sequence models where tokens are predicted sequentially based on past context. Although RNNs were the model of choice for many years, Transformers are now the dominant model in such applications that contain data that is inherently discrete. We show that alternative models to Transformers can still be competitive in these settings. By simply taking a strong Transformer baseline [8] and replacing the self-attention layers, S4 substantially closes the gap to Transformers (within 0.5 ppl), setting SOTA for attention-free models by over 2 ppl.

7.4 Additional Capabilities of S4

While Section 7.3 focused on raw performance metrics (e.g. classification accuracy or regression error) on established tasks, several of the tasks also demonstrate unique capabilities of SSMs.

Table 7.7: (**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. (throughput)
Transformer [8]	247M	20.51	0.8K (1×)
GLU CNN [38]	229M	37.2	-
AWD-QRNN [137]	151M	33.0	-
LSTM + Cache + Hebbian + MbPA [169]	-	29.2	-
TrellisNet [12]	180M	29.19	-
Dynamic Convolution [227]	255M	25.0	-
TaLK Convolution [126]	240M	23.3	-
S4	249M	20.95	48K (60×)

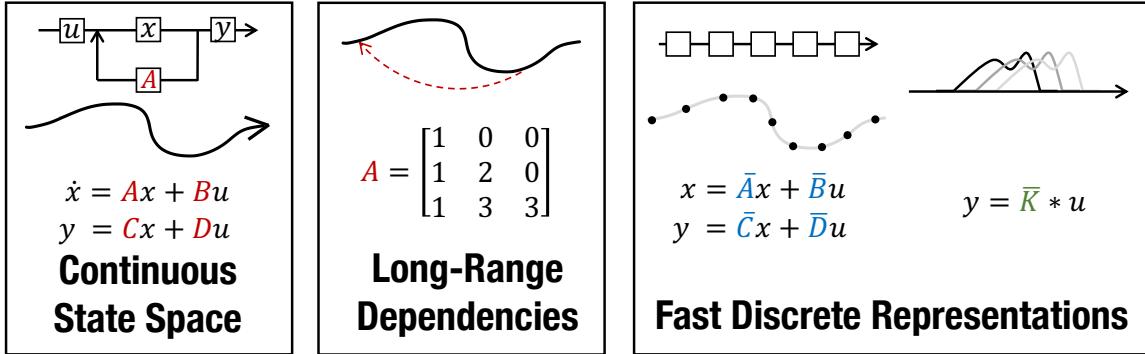


Figure 7.3: (**Properties of S4.**) (*Left*) Continuous SSMs map an input signal $u(t)$ to output $y(t)$ through a latent state $x(t)$. (*Center*) The HIPPO theory on continuous-time memorization derives special SSM matrices that can model LRDs mathematically and empirically. (*Right*) SSMs can be computed either as a recurrence (left) or convolution (right). Utilizing different representations of its parameters (red, blue, green), allows S4 to handle a wide range of tasks, be efficient at both training and inference, and excel at modeling long sequences.

We highlight particular capabilities induced by the theoretical properties of S4, including its three representations (continuous-time, recurrent, and convolutional) (Chapter 2) and its use of principled memory representations (Chapter 4). Figure 7.3 recaps these properties visually.

7.4.1 Continuous-time: Sampling Rate Adaptation

Resolution Change

As a continuous-time model, S4 automatically adapts to data sampled at different rates, a challenging setting for time series with a dedicated line of work [176, 40, 174]. This setting

can be a real-world problem; for example, deployed healthcare models can be tested on EEG signals that are sampled at a different rates [180, 185]. Similarly, speech waveforms are a continuous signal that are sampled at numerous standard frequencies such as 16kHz, 44.1kHz, 48kHz, or 88.2kHz (Figure 7.4).

Without re-training, S4 preserves its performance on Speech Commands when the data is sampled at $0.5 \times$ the training frequency (Table 7.3), simply by changing its internal step size Δ (Section 2.4.1). Table 7.3a includes other continuous-time models that have this capability, but unfortunately cannot achieve good performance in the first place because of the challenging nature of this dataset. On the other hand, Table 7.3b has much stronger CNN baselines for speech, which cannot adapt to different sampling rates because their discrete CNN kernels are attuned to a specific resolution.

We additionally point to other experiments in this thesis with similar highlights:

- The experiment in Section 4.4.3 shows even stronger timescale robustness properties when using the LSI version of HIPPO-LegS as a pure recurrence.
- Chapter 9 shows that this property can be adapted to other modalities such as images and videos, which can be interpreted as higher-dimensional continuous signals.

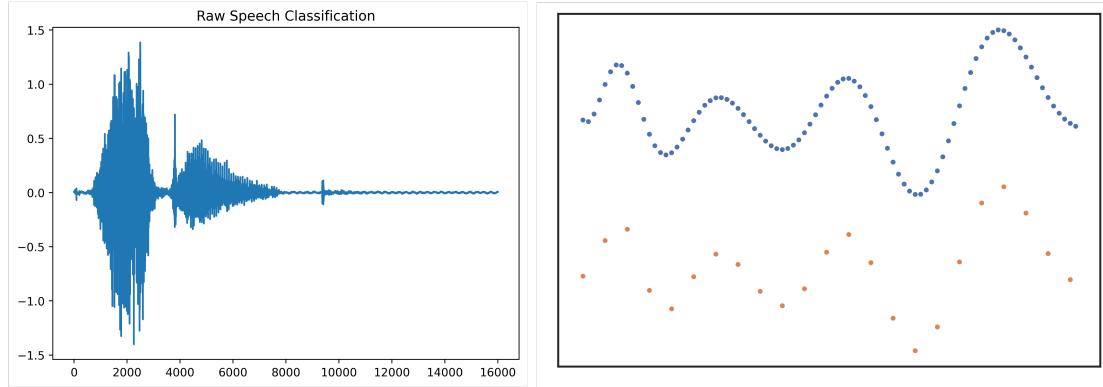


Figure 7.4: (*Left*) Visualization of a speech waveform from the Speech Commands dataset. (*Right*) Visualization of a continuous signal resampled at different rates. For typical sequence models, deploying on a different rate than it was trained on (e.g. *Blue* \rightarrow *Orange*) destroys performance.

7.4.2 Recurrent: Efficient Inference

A prominent limitation of autoregressive models is inference speed (e.g. generation), since they require a pass over the full context for every new sample. Several methods have been specifically crafted to overcome this limitation such as the Linear Transformer, a hybrid

Table 7.8: SSSMs: S4 (with Algorithm 1) is asymptotically more efficient than a naive SSM.

Dim.	TRAINING STEP (MS)			MEMORY ALLOC. (MB)		
	128	256	512	128	256	512
SSSM	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 7.9: Benchmarks vs. Transformers

	LENGTH 1024		LENGTH 4096	
	Speed	Mem.	Speed	Mem.
Transformer	1×	1×	1×	1×
Performer	1.23×	<u>0.43</u> ×	3.79×	<u>0.086</u> ×
Linear Trans.	1.58 ×	0.37 ×	5.35 ×	0.067 ×
S4	1.58 ×	<u>0.43</u> ×	<u>5.19</u> ×	0.091×

Transformer/RNN that switches to a recurrent view at inference time for speed [154], or the recent RWKV [158].

As a stateful model, SSMs automatically have this ability. By switching to its recurrent representation (Section 2.3.2), S4 requires *constant memory and computation* per time step – in contrast to standard autoregressive models which scale in the context length. On both CIFAR-10 and WikiText-103, we report the throughput of various models at generation time, with S4 around 60× faster than a vanilla Transformer on both tasks (details in Appendix G.2.1).

We also note that Chapter 8 also displays an application to audio waveform generation leveraging S4’s fast autoregressive generation.

7.4.3 Convolutional: Efficient Training

We benchmark that S4 can be trained quickly and efficiently, both compared to a naive SS(S)M, as well as efficient Transformer variants designed for long-range sequence modeling. As discussed in Chapter 3, S4 is theoretically much more efficient than general SSMs, and Table 7.8 confirms that the S4 is orders of magnitude more speed- and memory-efficient for practical layer sizes. In fact, S4’s speed and memory use is competitive with the most efficient Transformer variants benchmarked by Tay et al. [202]—Linear Transformer [104] and Performer [30]—in a parameter-matched setting (Table 7.9, following the protocol of Tay et al. [202]). (Details in Appendix G.2.2.)

7.4.4 Long-Range Dependencies

As described in Chapter 6, S4 uses a principled approach to address LRDs based on the HIPPO theory of continuous memorization. We note that almost all of the tasks in Section 7.3 are also considered LRD tasks in the sequence modeling literature, and many were in fact invented primarily to stress tests LRDS (e.g. sequential MNIST for RNNs [5], and

LRA for Transformers [202]). These datasets—which have sequences ranging from length roughly 1000 to 16000—are all very challenging for standard RNN, CNN, and Transformer models, and S4 set significant state-of-the-art improvements, serving as empirical validation that S4 handles LRDs efficiently.

7.5 Ablations of the S4 and HIPPO Theory

Our development of S4 specified many details in its parameterization and many variants of the methods. We perform an in-depth study of all of these components, and validate that all aspects of the theoretical results can be important empirically. A central theme of these ablations is the importance of instantiating (or initializing) all SSM parameters ($\Delta, \mathbf{A}, \mathbf{B}, \mathbf{C}$) appropriately, especially the state matrix \mathbf{A} , which otherwise causes the modeling issues described throughout this thesis (e.g. Chapter 1, Challenge 3).

- Section 7.5.1 gives an overview of our general protocol for ablations, which covers several domains, tasks, and sequence lengths.
- Section 7.5.2 looks at the effect of choosing the timescale Δ properly.
- Section 7.5.3 compares two minor choices in parameterizing SSM described in Section 3.2.
- Section 7.5.4 compares initializations of the critical \mathbf{A} matrix in S4-Diag. The ablations show that simple variants can substantially degrade performance, underscoring the importance of choosing \mathbf{A} carefully (Section 6.3).
- Section 7.5.5 analyzes the trainability of core parameters.
- Section 7.5.6 discusses the empirical differences between S4-Diag and S4-DPLR.
- Section 7.5.7 shows that for the full (DPLR) S4 method, incorporating HIPPO theory (Chapter 6) can have a noticeable empirical benefit, validating the importance of incorporating principled state representations.
- Finally, Section 7.5.8 compares S4 variants on a different set of synthetic tasks to examine the empirical strengths of different HIPPO methods.

7.5.1 Ablations Protocol

In order to study the effects of different S4 and S4D variants in a controlled setting, we propose the following protocol.

Datasets

We focus on three datasets covering a varied range of data modalities (image pixels, biosignal time series, audio waveforms), sequence lengths (1K, 4K, 16K), and tasks (classification and regression with bidirectional and causal models). These datasets were all used in the general results in Section 7.3, summarized below for convenience.

- **Sequential CIFAR (sCIFAR).** CIFAR-10 images are flattened into a sequence of length 1024, and a bidirectional sequence model is used to perform 10-way classification.
- **BIDMC Vital Signs.** EKG and PPG signals of length 4000 are used to predict respiratory rate (RR), heart rate (HR), and blood oxygen saturation (SpO2). We focus on SpO2 in this study.
- **Speech Commands (SC).** A 1-second raw audio waveform comprising 16000 samples is used for 35-way spoken word classification. We use an autoregressive (AR) model to vary the setting; this causal setting more closely imitates autoregressive speech generation, an interesting application of S4 studied in Chapter 8.

Training Protocol

Instead of the larger models in Section 7.3, we use a much smaller model and simple generic training protocol to study ablations.

- The architecture has 4 layers and hidden dimension $H = 128$, resulting in about $100K$ parameters.
- All results are averaged over 2 or 3 seeds.
- The \mathbf{A} and \mathbf{B} parameters were tied across the H SSM copies (Section 7.2.2). This means that when comparing Diagonal vs. DPLR parameterizations or frozen vs. trainable parameters, the effects of parameter count is negligible (at most 1% of trainable parameters).

- All models use learning rate 0.004, 0.01 weight decay, and no other regularization or data augmentation.
- For the classification tasks (sCIFAR and SC), we use a cosine scheduler with 1 epoch warmup and decaying to 0. For the regression task (BIDMC), we use a multistep scheduler following [178, 74].
- Reported results are all best validation accuracy/MSE instead of the test metric corresponding to best validation metric.

7.5.2 O-SSM Normalization Theory: Initialization of Δ and C

Δ Initialization

On a very challenging task such as LRA’s Path-X (Section 7.3.4), the theory of timescales in Section 5.4 can make a large difference. Figure 7.5 illustrates the importance of setting Δ correctly. Instead of the standard initialization of $(\Delta_{min}, \Delta_{max}) = (0.001, 0.1)$, the best results are obtained by lowering the initialization of Δ by a factor of 10 in accordance with known length of dependencies in the Path-X task.

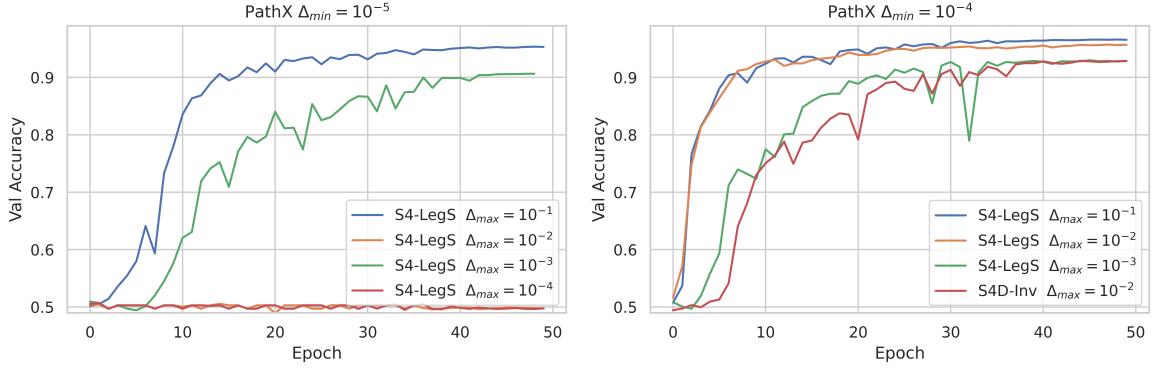


Figure 7.5: (**Timescale ablation on Path-X; validation curves.**) (*Left*) Setting Δ_{min} too small can solve the task, but is inconsistent. (*Right*) A good setting of Δ_{min} can consistently solve the task. Note that the timescale of each feature is up to $\frac{1}{\Delta_{min}} = 10^4$, which is on the order of (but not exceeding) the length of the task $L = 16384$. Empirically, performance is best when spreading out the range of Δ with a larger Δ_{max} that covers a wider range of timescales and can potentially learn features at different resolutions, which are combined by a multi-layer deep neural network.

C Initialization

We validate the theory of normalization in Section 5.4, which predicts that for a properly normalized TO-SSM, the projection parameter C should be initialized with unit variance,

Table 7.10: Ablation of the initialization of \mathbf{C} for S4(-LegS) on classification datasets.

Init. std. σ of \mathbf{C}	0.01	0.1	1.0	10.0	100.0
Sequential CIFAR	85.91 (0.41)	86.33 (0.01)	86.49 (0.51)	84.40 (0.16)	82.05 (0.61)
Speech Commands	90.27 (0.31)	90.00 (0.11)	90.67 (0.19)	90.30 (0.36)	89.98 (0.51)

in contrast to standard initializations for deep neural networks which normalize by a factor related to the size of N . Note that standard initializations such as Xavier initialization [66] would initialize the variance of \mathbf{C} to $O(1/N)$ (in this case $N = 64$). Table 7.10 shows classification results on datasets Sequential CIFAR (sCIFAR) and Speech Commands (SC). Results show that using standard deviation 1.0 for \mathbf{C} is slightly better than alternatives, although the difference is usually minor.

7.5.3 SSM Parameterization

Given the same diagonal SSM matrices \mathbf{A}, \mathbf{B} , there are many variants of how to parameterize the matrices and compute the SSM kernel described in Section 3.4. We ablate some choices described in Table 3.1. Results are in Table 7.11, and show that:

- (i) Different discretizations (Section 3.4.1) do not make a noticeable difference. In particular, we can always use the bilinear discretization to match the parameterizations of S4D and S4.
- (ii) Different constraints on the real part of \mathbf{A} (Section 3.4.2) are similar. Although unrestricting it may seem to perform slightly better, we always constrain the eigenvalues of \mathbf{A} to be negative, ensuring stability in recurrent mode (Section 3.4.2).

Therefore we can preserve a parameterization ensuring that S4D exactly matches S4, in the sense that they would compute the same kernel when the low-rank portion of S4’s DPLR representation is set to 0 (Section 3.5). This allows controlled comparisons of the critical state matrix \mathbf{A} for the remainder of the ablations in this section.

Remark 7.2. *More ablations and analysis for constraining the real part of \mathbf{A} , and enforcing the Hurwitz condition in general (Section 3.3.3), are shown in Chapter 8.*

7.5.4 S4D Initialization Ablations

The original S4 model proposed a specific formula for the \mathbf{A} matrix based on HIPPO-LegS, and our main diagonal version S4D-LegS also uses specific matrix based on it (Section 6.3).

Table 7.11: Ablations of different parameterizations of diagonal SSMs using S4D-Inv.

Discretization	Real part of \mathbf{A}	SCIFAR	SC (AR)	BIDMC (SpO2)
Bilinear	exp	85.20	89.52	0.1193
	ReLU	85.06	90.22	0.1172
	-	85.35	90.58	0.1102
ZOH	exp	85.02	89.93	0.1303
	ReLU	84.98	90.03	0.1232
	-	85.15	90.19	0.1289

Our other proposed variants S4D-Inv and S4D-Lin also define precise formulas for the initialization of the \mathbf{A} matrix (6.5). Empirically, results in Section 7.3 show that they generally all perform well. This raises the question of whether the initialization of the \mathbf{A} still needs to be so precise, despite the large simplifications from the full HIPPO structure. We perform several natural ablations on these initializations, showing that even simple variations of the precise formula can degrade performance. These results are summarized in Table 7.12a (*Left*).

Imaginary Part Scaling Factor

The scaling rules for the imaginary parts of S4D-Inv and S4D-Lin are simple polynomial laws, but how is the constant factor chosen and how important is it? These constants are based on approximations to HIPPO methods (e.g. Conjecture 6.11). Note that the range of imaginary components for S4D-Inv and S4D-Lin are quite different (Figure 6.2); the largest imaginary part is $\frac{N^2}{\pi}$ for S4D-Inv and πN for S4D-Lin.

We consider scaling all imaginary parts by a constant factor of 0.01 or 100.0 to investigate whether the constant matters. Note that this preserves the overall shape of the basis functions (Figure 3.1, dashed lines) and simply changes the frequencies (because the real part \mathbf{A}_{Re} controls the decay), and it is not obvious that this should degrade performance. However, both changes substantially reduce the performance of S4D in all settings.

Randomly Initialized Imaginary Part

Next, we consider choosing the imaginary parts randomly. For S4D-Inv, we keep the real parts equal to $-\frac{1}{2}$ and set each imaginary component to

$$\mathbf{A}_n = -\frac{1}{2} + i \frac{N}{\pi} \left(\frac{N}{2u+1} - 1 \right) \quad u \sim N \cdot \mathcal{U}[0, 1] \quad (7.1)$$

where \mathcal{U} denotes the uniform distribution. Note that when u is equally spaced in $[0, 1]$ instead of uniformly random, this exactly recovers S4D-Inv (6.5), so this is a sensible random approximation to it.

Similarly, we consider a variant of S4D-Lin

$$\mathbf{A}_n = -\frac{1}{2} + i\pi u N \quad u \sim N \cdot \mathcal{U}[0, 1] \quad (7.2)$$

that is equal to equation (6.6) when u is equally spaced instead of random.

Table 7.12a (*Random Imag*) shows that this small change causes minor degradation in performance. We additionally note that the randomly initialized imaginary ablation can be interpreted as follows. Figure 6.2 shows the asymptotics of the imaginary parts of SSM matrices, where the imaginary parts of the eigenvalues correspond to y-values corresponding to uniformly spaced nodes on the x-axis. This ablation then replaces the uniform spacing on the x-axis with uniformly random x values.

Randomly Initialized Real Part

We considering initializing the real part of each eigenvalue as $-\mathcal{U}[0, 1]$ instead of fixing them to $-\frac{1}{2}$. Table 7.12a (Left, *Random Real*) shows that this also causes minor but consistent degradation in performance on the ablation datasets. Finally, we also consider randomizing both real and imaginary parts, which degrades performance even further.

Ablation: Other S4D matrices. Other simple variants of initializations show that it is not just the range of the eigenvalues but the actual distribution that is important (Figure 6.2). We define two variants S4D-Inv2 and S4D-Quad, both with real part $-\frac{1}{2}$ and imaginary part satisfying the same maximum value as Conjecture 6.11. The S4D-Inv2 initialization uses the same formula as S4D-Inv, but replaces a $2n+1$ in the denominator with $n+1$. The S4D-Quad initialization uses a polynomial law with power 2 instead of -1

Table 7.12: (*Initialization and Trainability ablations*)

Ablation	sCIFAR	SC (AR)	BIDMC
S4D-Lin	85.12	90.66	0.128
Scale 0.01	-7.27	-1.92	+0.040
Scale 100	-7.91	-4.04	+0.077
Random Imag	-0.42	-3.08	-0.001
Random Real	-0.73	-0.87	+0.011
Random Both	-1.28	-5.88	+0.007
S4D-Inv	84.79	90.27	0.114
Scale 0.01	-5.03	-0.08	+0.028
Scale 100	-7.77	-52.31	+0.034
Random Imag	-0.29	-0.52	+0.010
Random Real	0.12	-2.18	+0.032
Random Both	-1.55	-0.55	+0.024
S4D-Inv2	-2.62	-39.84	+0.005
S4D-Quad	-1.83	-0.62	+0.024
S4D-Random	-6.32	-1.95	+0.034
S4D-Real	-5.45	-10.17	+0.066

	sCIFAR		SC (AR)		BIDMC
Frozen (\mathbf{A}, \mathbf{B})	Acc (first)	Acc (best)	Acc (first)	Acc (best)	RMSE (best)
S4-LegS	53.63	86.19	33.87	85.33	0.1049
S4-LegT	54.76	86.30	8.77	57.35	0.1106
S4-FouT	55.28	86.05	9.27	69.57	0.1072
S4-LegS+FouT	54.38	86.53	34.06	83.37	0.0887
S4D-LegS	50.87	84.81	22.76	77.18	0.0960
S4D-Inv	53.19	84.40	18.49	76.53	0.0995
S4D-Lin	51.75	84.96	19.09	75.58	0.0935

	Trainable (\mathbf{A}, \mathbf{B})				
	54.23	86.29	62.19	90.68	0.1033
S4-LegS	55.16	86.12	55.86	90.42	0.1146
S4-LegT	55.89	85.93	60.56	90.83	0.1136
S4-FouT	55.00	86.18	61.76	91.01	0.0970
S4-LegS+FouT	50.41	85.64	47.54	88.47	0.1148
S4D-LegS	53.42	84.59	45.73	89.69	0.1132
S4D-Inv	52.23	85.75	47.68	89.56	0.1032
S4D-Lin					

- (a) Ablations of the init. of the diagonal (b) Results for all S4 and S4D methods on the ablation \mathbf{A} matrix in S4D. Very simple changes that datasets, when the (\mathbf{A}, \mathbf{B}) matrices are either frozen (*Top*) or largely preserve the structure of the diagno- trained (*Bottom*). Diagonal state matrices are highly comal eigenvalues all degrade performance. petitive with full DPLR versions in general.

(S4D-Inv) or 1 (S4D-Lin).

$$(\mathbf{S4D-Inv2}) \quad \mathbf{A}_n = -\frac{1}{2} + i \frac{N}{\pi} \left(\frac{N}{n+1} - 1 \right) \quad (7.3)$$

$$(\mathbf{S4D-Quad}) \quad \mathbf{A}_n = \frac{1}{\pi} (1 + 2n)^2 \quad (7.4)$$

We also include two additional methods here that are not based on the proposed S4D-Inv or S4D-Lin methods. First, S4D-Rand uses a randomly initialized diagonal \mathbf{A} , and validates that it performs poorly, in line with earlier findings [71, 76]. Second, S4D-Real uses a particular real initialization with $\mathbf{A}_n = -(n+1)$. This is the exact same spectrum as the original LegS matrix (Equation (5.1)), which validates that it is not just the diagonalization that matters, highlighting the limitations of Proposition 3.4 (see Section 3.5.1).

7.5.5 Trainability of (\mathbf{A}, \mathbf{B}) Matrices

Table 7.12b shows the performance of all S4D and S4 variants on the ablations datasets. We observe that freezing the matrices performs comparably to training them on sCIFAR and BIDMC, but is substantially worse on SC. We hypothesize that this results from Δ being poorly initialized for SC, so that at initialization models do not have context over

the entire sequence, and training \mathbf{A} and \mathbf{B} helps adjust for this. As further evidence, the *finite window methods* S4-LegT and S4-FouT (Section 5.3) have the most limited context and suffer the most when \mathbf{A} is frozen.

We note that because of the parameter-tying (Section 7.5.1), learning A adds only $O(N)$ parameters, adding less than 1% parameter count compared to the base models with $O(HN)$ parameters. Thus the effects of trainability are not a result of having more parameters.

7.5.6 S4D vs. S4

Table 7.12b also shows that the more general DPLR SSMs are often slightly better than the diagonal version throughout the entire training curve, at least for these small ablation models. We report the validation accuracy after 1 epoch of training on sCIFAR and SC to illustrate this phenomenon. Note that this is also not a consequence of having more parameters because we use weight-tying (Section 7.5.1).

Large Models on Ablation Datasets

While this section focused on controlled ablations with small models, full results for larger models were presented in Table 7.3b for SC and Table 7.2 for BIDMC. Compared to this section, the model size and regularization strength is simply increased.

Remark 7.3. *Large model results for sCIFAR comparing the S4 and S4D variants are not included in this thesis, but can be found in the original paper [73, Table 9].*

Overall, our experimental study in Section 7.3 shows that S4D has strong performance in a wide variety of domains and tasks, and is generally competitive with S4 on all tasks while significantly outperforming all baselines prior to S4.

7.5.7 DPLR Ablations: The Effect of HIPPO

A critical motivation of S4 was the use of the HIPPO matrices to initialize an SSM. We consider several simplifications of S4 to ablate the importance of each of these components, including:

- (i) How important is the HIPPO(-LegS) initialization?
- (ii) How important is training the SSM on top of HIPPO?

(iii) Are the benefits of S4 captured by the DPLR parameterization without HIPPO?

As a simple testbed, all experiments in this section were performed on the sequential CIFAR-10 task, which we found transferred well to other settings. Models were constrained to at most 100K trainable parameters and trained with a simple plateau learning rate scheduler and no regularization.

Unconstrained SSMs

We first investigate generic SSMs with various initializations. We consider a random Gaussian initialization (with variance scaled down until it did not NaN), and the HIPPO initialization. We also consider a random diagonal Gaussian matrix as a potential structured method; parameterizing \mathbf{A} as a diagonal matrix would allow substantial speedups without going through the complexity of S4’s NPLR parameterization. We consider both freezing the \mathbf{A} matrix and training it.

Figure 7.6 shows both training and validation curves, from which we can make several observations. First, training the SSM improved all methods, particularly the randomly initialized ones. For all methods, training the SSM led to improvements in both training and validation curves.

Second, a large generalization gap exists between the initializations. In particular, note that when \mathbf{A} is trained, all initializations are able to reach perfect training accuracy. However, their validation accuracies are separated by over 15%.

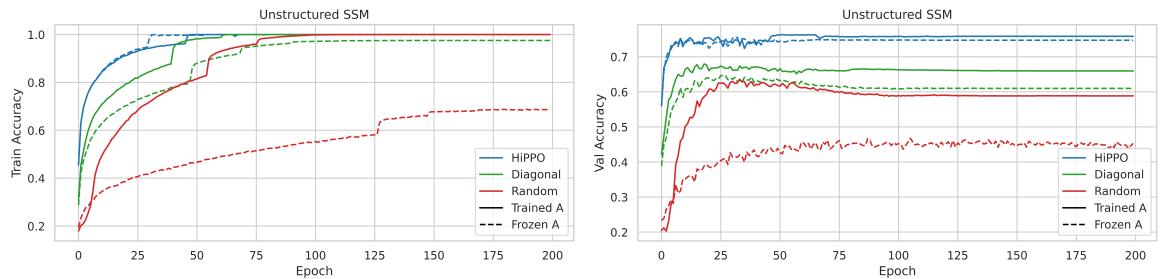


Figure 7.6: CIFAR-10 classification with unconstrained, real-valued SSMs with various initializations. (Left) Train accuracy. (Right) Validation accuracy.

DPLR SSMs

The previous experiment validates the importance of HIPPO in SSMs. This was the main motivation of the DPLR parameterization in S4, which utilizes structure of the HIPPO

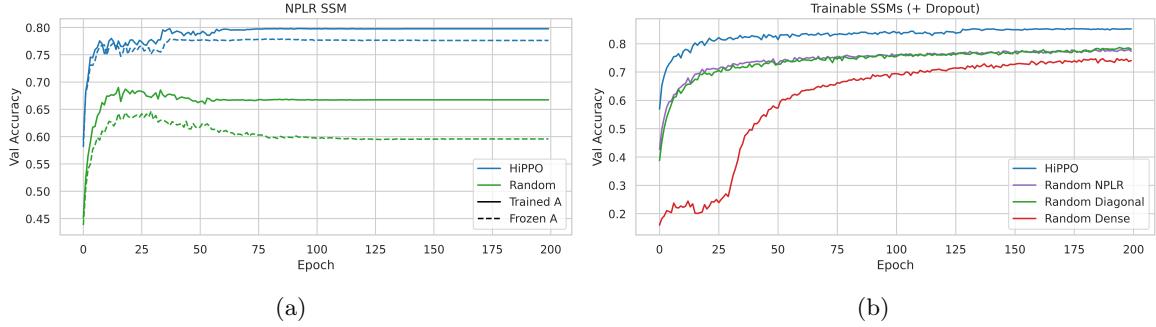


Figure 7.7: CIFAR-10 validation accuracy of SSMs with different initializations and parameterizations. (*Left*) DPLR parameterization with random versus HIPPO initialization. (*Right*) All methods considered in this section, including minor Dropout regularization. S4 achieves SOTA accuracy on sequential CIFAR-10 with just 100K parameters.

matrix (6.1) to make it computationally feasible in an SSM. Figure 7.7a shows that random DPLR matrices still do not perform well, which validates that S4’s effectiveness primarily comes from the HIPPO initialization, not the DPLR parameterization.

Finally, Figure 7.7b considers the main ablations considered in this section (with trainable SSMs) and adds minor regularization. With 0.1 dropout [193], the same trends still hold, and the HIPPO initialization—in other words, the full S4 method—achieves 84.27% test accuracy with just 100K parameters.

Remark 7.4. *We note that the importance of using HIPPO (and principled initializations in general) seems to go away with larger-scale models and datasets. This is perhaps intuitive given that more training means more opportunity to attenuate the effect of bad initializations, although such principles in the theory of deep learning optimization is still not well understood. An example of this is the S4(D)-Rand ablation on LRA (Table 7.4), which is competitive overall, although is still worse on several tasks.*

7.5.8 Comparisons Between HIPPO Methods

Finally, we study some empirical tradeoffs of the S4 variants from Chapter 5. We compare several S4 variants which are TO-SSMs (requiring DPLR form; c.f. Theorem 6.7), as well as to the simpler S4D variants that are not technically orthogonal SSMs. Corresponding to the intuition in Section 5.5, we hypothesize that

- S4-LegS excels at sparse memorization tasks because it represents very smooth convolution kernels that memorize the input against an infinitely-long measure (Corollary 5.10,

Figure 5.1). Conversely, it is less appropriate at short-range tasks with dense information because it smooths out the signal.

- S4-FouT excels at dense memorization tasks because it can represent spike functions that pick out past elements at particular ranges (Section 5.3.2). However, it is less appropriate at very long range tasks because it represents a finite (local) window.
- Δ can be initialized precisely based on known time dependencies in a given task to improve performance.

Theory: Function Reconstruction, Timescales, Normalization

S4-FouT (and S4-LegT) are orthogonal SSMs with respect to a uniform measure, that have closed form formulas allowing function reconstruction. We construct a synthetic **Reconstruction Task** to test if S4 variants can instead be *trained* to reconstruct (for a uniform measure) without these formulas.

The input is a white noise sequence $u \in \mathbb{R}^{4000}$. We use a single layer linear S4 model with state size $N = 256$ and $H = 256$ hidden units. Models are required to use their output at the last time step, a vector $y_{4000} \in \mathbb{R}^{256}$, to reconstruct the last 1000 elements of the input with a linear probe. Concretely, the loss function is to minimize $\|u_{3000:4000} - \mathbf{W}y_{4000}\|_2^2$, where $\mathbf{W} \in \mathbb{R}^{1000 \times 256}$ is a learned matrix.

Figure 7.8 shows that S4-LegT and S4-FouT, the methods that theoretically reconstruct against a uniform measure, are far better than other methods. A natural question is whether other SSMs that do not satisfy the HIPPO theory can still perform well. To test this, we include the diagonal S4D variants, which are simpler SSM methods that are not O-SSMs but generally perform well empirically. However, they do not learn the right function on this task. We also include a method **S4-(LegS/FouT)** which combines both LegS and FouT measures by simply initializing half of the SSM kernels to each. Despite having fewer S4-FouT kernels, this still performs as well as the pure S4-FouT initialization.

Memorization: the Delay (continuous copying) Task

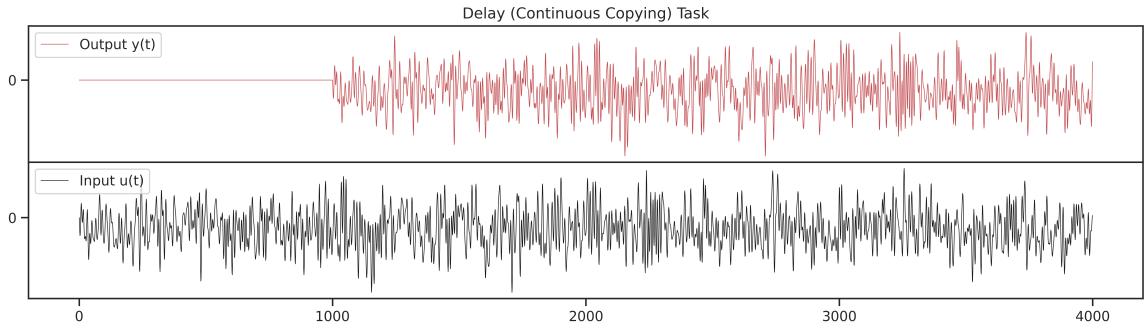
Next, we study how the synthetic reconstruction ability transfers to other tasks. We construct a task that requires models to learn a sequence-to-sequence map whose output is the input lagged by a fixed time period (Figure 7.9a).

	(Δ min, Δ max)		(Δ min, Δ max)	
	(1e-3, 2e-3)	(2e-3, 2e-3)	(1e-3, 1e-1)	(2e-3, 1e-1)
S4-LegS	-6.2581	-6.6328	-3.5505	-3.3017
S4-LegT	-7.4987	-8.1056	-2.9729	-2.6557
S4-FouT	-7.4889	-8.3296	-3.0062	-2.6976
S4-(LegS/FouT)	-7.4992	-8.3162	-3.4784	-3.2628
S4D-LegS	-6.1528	-6.184	-3.8773	-3.6317
S4D-Inv	-5.9362	-6.0986	-4.1402	-3.7912
S4D-Lin	-7.1233	-6.6483	-3.973	-3.5991
S4D-(Inv/Lin)	-6.839	-6.705	-4.325	-3.8389

Figure 7.8: Log-MSE after training on the Reconstruction Task. (*Left*) When the timescales Δ are set appropriately for this task, the methods that theoretically reconstruct against a uniform measure (LegT and FouT) are much better than alternatives, achieving MSE more orders of magnitude lower than other SSM initializations. (*Right*) Interestingly, when the timescales Δ are not set correctly, these methods (LegT and FouT) actually perform worst and S4D variants perform best.

Delay (Continuous Copying) Task The Delay Task consists of input-output pairs where the input is a white noise signal of length 4000 bandlimited to 1000 Hz. The output is the same signal shifted by 1000 steps (Figure 7.9a). We use single layer linear SSMs with $H = 4$ hidden units and state size $N = 1024$. Models are trained with the Adam optimizer with learning rate 0.001 for 20 epochs.

Results For recurrent models, this task can be interpreted as requiring models to maintain a memory buffer that continually remembers the latest elements it sees. This capability was the original motivation for the Legendre Memory Unit [218], the predecessor to HIPPO-LegT, which was explicitly designed to solve this task because it can encode a spike kernel (Figure 5.3). In Figure 7.9b, we see that our new S4-FouT actually outperforms S4-LegT, which both outperform all other methods when the timescale Δ is set correctly. We note that this task with a lag of just 1000 time steps is too hard for baselines such as an LSTM and Transformer, which empirically did not learn better than random guessing (RMSE 0.43) and are not reported.



(a) (**Delay Task**) Models perform a mapping from $\mathbb{R}^{4000} \rightarrow \mathbb{R}^{4000}$ where the target output is lagged by 1000 steps, with error measured by RMSE. The input is a white noise signal bandlimited to 1000Hz. We use single layer SSMs with state size $N = 1024$.

$(\Delta_{\min}, \Delta_{\max})$	Frozen (A, B)	Trainable (A, B)
(5e-4, 5e-4)	0.2891	0.2832
(1e-3, 1e-3)	0.1471	0.1414
(2e-3, 2e-3)	0.0584	0.0078
(4e-3, 4e-3)	0.4227	0.1262
(8e-3, 8e-3)	0.4330	0.2928
(5e-4, 1e-1)	0.2048	0.1537
(1e-3, 1e-1)	0.2017	0.1474
(2e-3, 1e-1)	0.3234	0.2262
(4e-3, 1e-1)	0.4313	0.3417
(8e-3, 1e-1)	0.4330	0.4026

	$(\Delta_{\min}, \Delta_{\max}) = (1e-3, 1e-1)$		$(\Delta_{\min}, \Delta_{\max}) = (2e-3, 2e-3)$	
	Frozen (A, B)	Trainable (A, B)	Frozen (A, B)	Trainable (A, B)
S4-LegS	0.3072	0.0379	0.2369	0.0130
S4-LegT	0.2599	0.1204	0.0226	0.0129
S4-FouT	0.2151	0.1474	0.0304	0.0078
S4-LegS+FouT	0.1804	0.0309	0.0250	0.0080
S4D-LegS	0.1378	0.0337	0.0466	0.0140
S4D-Inv	0.1489	0.0243	0.0605	0.0186
S4D-Lin	0.1876	0.1653	0.0421	0.0144

(b) (**RMSE**) (*Left*) Setting Δ appropriately makes a large difference. For FouT (A, B), which encode *finite window* basis functions (Figure 5.1), the model can see a history of length up to $\frac{2}{\Delta}$. For example, setting Δ too large means the model cannot see 1000 steps in the past, and performs at chance. Performance is best at the theoretically optimal value of $\Delta = 2 \cdot 10^{-3}$ which can encode a spike kernel at distance exactly 1000 steps (Corollary 5.17). (*Right*) When Δ is set optimally, the S4-FouT method is the best SSM as the theory predicts. When Δ is not set optimally, other methods perform better, including the simple diagonal methods which are not orthogonal SSMs.

Figure 7.9: (**Delay Task**.) A synthetic memorization task: visualization (*Top*, (a)) and results (*Bottom*, (b)).

Chapter 8

SaShiMi: S4 for Audio Waveform Generation

We present an application of S4 as a black box inside a new neural network architecture. The central contribution of this chapter is showing that SSNNs are a strong alternative to conventional architectures for modeling audio waveforms, with favorable tradeoffs in training speed, generation speed, sample efficiency, and audio quality. Audio examples can be found at <https://hazyresearch.stanford.edu/sashimi-examples>.

8.1 Introduction

Generative modeling of raw audio *waveforms* is a challenging frontier for machine learning due to their high-dimensionality—waveforms contain tens of thousands of timesteps per second and exhibit long-range behavior at multiple timescales. A key problem is developing architectures for modeling waveforms with the following properties:

1. *Globally coherent* generation, which requires modeling unbounded contexts with long-range dependencies.
2. *Computational efficiency* through parallel training, and fast autoregressive and non-autoregressive inference.
3. *Sample efficiency* through a model with inductive biases well suited to high-rate waveform data.

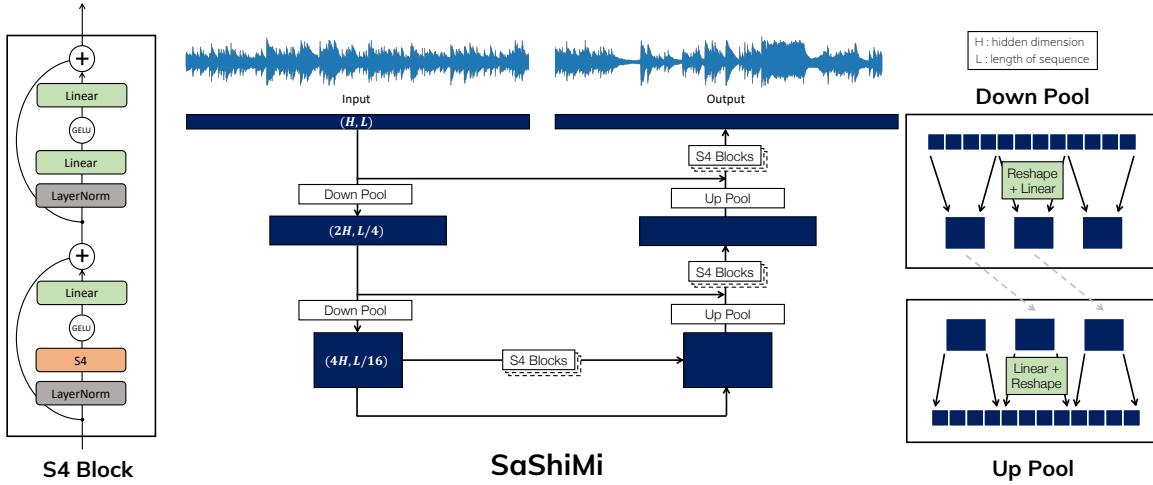


Figure 8.1: SASHIMI consists of a simple repeated block combined with a multiscale architecture. (*Left*) The basic S4 block is composed of an S4 layer combined with standard pointwise linear functions, nonlinearities, and residual connections. (*Center*) Dark blue rectangles illustrate the shape of inputs. The input is progressively transformed to shorter and wider sequences through pooling layers, then transformed back with stacks of S4 blocks. Longer range residual connections are included to help propagate signal through the network. (*Right*) Pooling layers are position-wise linear transformations with shifts to ensure causality.

Among the many training methods for waveform generation, autoregressive (AR) modeling is a fundamentally important approach. AR models learn the distribution of future variables conditioned on past observations, and are central to recent advances in machine learning for language and image generation [18, 171, 16]. With AR models, computing the exact likelihood is tractable, which makes them simple to train, and lends them to applications such as lossless compression [110] and posterior sampling [96]. When generating, they can condition on arbitrary amounts of past context to sample sequences of unbounded length—potentially even longer than contexts observed during training. Moreover, architectural developments in AR waveform modeling can have a cascading effect on audio generation more broadly. For example, WaveNet—the earliest such architecture [148]—remains a central component of state-of-the-art approaches for text-to-speech (TTS) [122], unconditional generation [118], and non-autoregressive (non-AR) generation [112].

Despite notable progress in AR modeling of (relatively) short sequences found in domains such as natural language (e.g. 1K tokens), it is still an open challenge to develop architectures that are effective for the much longer sequence lengths of audio waveforms (e.g. 1M samples). Past attempts have tailored standard sequence modeling approaches like CNNs [148], RNNs [135], and Transformers [27] to fit the demands of AR waveform

modeling, but these approaches have limitations. For example, RNNs lack computational efficiency because they cannot be parallelized during training, while CNNs cannot achieve global coherence because they are fundamentally constrained by the size of their receptive field.

We introduce **SaShiMi**, a new architecture for modeling waveforms that yields state-of-the-art performance on unconditional audio generation benchmarks in both the AR and non-AR settings. SAShIMI is designed around SSMs (namely S4), which have a number of key features that make them ideal for modeling raw audio data. Concretely, S4:

1. Incorporates a principled approach to modeling long range dependencies with strong results on long sequence modeling, including raw audio classification.
2. Can be computed either as a CNN for efficient parallel training, or an RNN for fast autoregressive generation.
3. Is implicitly a continuous-time model, making it well suited to signals like waveforms.

Specifically, SAShIMI incorporates S4 into a UNet-like architecture [175] that uses pooling layers between blocks of residual S4 layers to capture hierarchical information across multiple resolutions. This is a common technique in neural network architectures such as standard CNNs and multi-scale RNNs, and provides empirical improvements in both performance and computational efficiency over isotropic stacked S4 layers.

For AR modeling in audio domains with unbounded sequence lengths (e.g. music), SAShIMI can train on much longer contexts than existing methods including WaveNet (sequences of length 128K vs 4K), while simultaneously having better test likelihood, faster training and inference, and fewer parameters. SAShIMI outperforms existing AR methods in modeling the data (> 0.15 bits better negative log-likelihoods), with substantial improvements (+0.4 points) in the musicality of long generated samples (16s) as measured by mean opinion scores. In unconditional speech generation, SAShIMI achieves superior global coherence compared to previous AR models on the difficult SC09 dataset both quantitatively (80% higher inception score) and qualitatively (2 \times higher audio quality and digit intelligibility opinion scores by human evaluators).

Finally, we validate that SAShIMI is a versatile backbone for non-AR architectures. Replacing the WaveNet backbone with SAShIMI in the state-of-the-art diffusion model DiffWave improves its quality, sample efficiency, and robustness to hyperparameters with no additional tuning.

8.2 Related Work

This work focuses primarily on the task of generating raw audio waveforms without conditioning information. Most past work on waveform generation involves conditioning on localized intermediate representations like spectrograms [187, 117, 165], linguistic features [148, 100, 15], or discrete audio codes [147, 46, 45, 118]. Such intermediaries provide copious information about the underlying content of a waveform, enabling generative models to produce globally-coherent waveforms while only modeling local structure.

In contrast, modeling waveforms in an unconditional fashion requires learning both local and global structure with a single model, and is thus more challenging. Past work in this setting can be categorized into AR approaches [148, 135, 27], where audio samples are generated one at a time given previous audio samples, and non-AR approaches [49, 112], where entire waveforms are generated in a single pass. While non-AR approaches tend to generate waveforms more efficiently, AR approaches have two key advantages. First, unlike non-AR approaches, they can generate waveforms of unbounded length. Second, they can tractably compute exact likelihoods, allowing them to be used for compression [110] and posterior sampling [96].

In addition to these two advantages, new architectures for AR modeling of audio have the potential to bring about a cascade of improvements in audio generation more broadly. For example, while the WaveNet architecture was originally developed for AR modeling (in both conditional and unconditional settings), it has since become a fundamental piece of infrastructure in numerous audio generation systems. For instance, WaveNet is commonly used to *vocode* intermediaries such as spectrograms [187] or discrete audio codes [147] into waveforms, often in the context of text-to-speech (TTS) systems. Additionally, it serves as the backbone for several families of non-AR generative models of audio in both the conditional and unconditional settings:

- (i) Distillation: Parallel WaveNet [146] and ClariNet [161] distill parallelizable flow models from a teacher WaveNet model.
- (ii) Likelihood-based flow models: WaveFlow [162], WaveGlow [165], and FloWaveNet [108] all use WaveNet as a core component of reversible flow architectures.
- (iii) Autoencoders: WaveNet Autoencoder [55] and WaveVAE [159], which use WaveNets in their encoders.
- (iv) Generative adversarial networks (GAN): Parallel WaveGAN [230] and GAN-TTS [15],

Table 8.1: Summary of music and speech datasets used for unconditional AR generation experiments.

CATEGORY	DATASET	TOTAL DURATION	CHUNK LENGTH	SAMPLING RATE	QUANTIZATION	SPLITS (TRAIN-VAL-TEST)
MUSIC	BEETHOVEN	10 HOURS	8s	16kHz	8-BIT LINEAR	MEHRI ET AL. [135]
MUSIC	YOUTUBEMIX	4 HOURS	8s	16kHz	8-BIT MU-LAW	88% – 6% – 6%
SPEECH	SC09	5.3 HOURS	1s	16kHz	8-BIT MU-LAW	WARDEN [222]

which use WaveNets in their discriminators.

- (v) Diffusion probabilistic models: WaveGrad [24] and DiffWave [112] learn a reversible noise diffusion process on top of dilated convolutional architectures.

In particular, we point out that DiffWave represents the state-of-the-art for unconditional waveform generation, and incorporates WaveNet as a black box.

Despite its prevalence, WaveNet is unable to model long-term structure beyond the length of its receptive field (up to 3s), and in practice, may even fail to leverage available information beyond a few tens of milliseconds [187]. Hence, we develop an alternative to WaveNet which can leverage unbounded context. We focus primarily on evaluating our proposed architecture SASHIMI in the fundamental AR setting, and additionally demonstrate that, like WaveNet, SASHIMI can also transfer to non-AR settings.

8.3 The SaShiMi Architecture

SASHIMI consists of two main components: S4 as the core sequence transformation, and a multi-scale SSNN architecture. Figure 8.1 illustrates the complete SASHIMI architecture.

First, S4 layers are the core component of our neural network architecture, to capture long context while being fast at both training and inference. We specifically use the DPLR form of S4 with the HIPPO-LegS initialization (Chapter 6). We note that building SASHIMI around S4 is similar in spirit to the use of linear oscillations for waveform generation in prior work, e.g. in the Neural Source Filter [221] or differentiable digital signal processing (DDSP) models [54]. SSMs like S4, and consequently SASHIMI, can be viewed as a generalization of models that rely on harmonic oscillation, and are able to directly learn the appropriate basis functions for raw waveforms.

In addition to the basic SSNN block in Section 7.2, we add a following “MLP” block, also called the **feed-forward network** in Transformers or the **inverted bottleneck layer** in CNNs [131].

Table 8.2: Results on AR modeling of Beethoven, a benchmark task from Mehri et al. [135]—SASHIMI outperforms all baselines while training faster.

MODEL	CONTEXT	NLL	@200K STEPS	@10 HOURS
SAMPLERNN*	1024	1.076	—	—
WAVENET*	4092	1.464	—	—
SAMPLERNN [†]	1024	1.125	1.125	1.125
WAVENET [†]	4092	1.032	1.088	1.352
SASHIMI	128000	0.946	1.007	1.095

*REPORTED IN MEHRI ET AL. [135]

[†]OUR REPLICATION

Second, SASHIMI connects stacks of S4 layers together in a simple multi-scale architecture that consolidates information from the raw input signal at multiple resolutions. The SASHIMI architecture consists of multiple tiers, with each tier composed of a stack of residual S4 blocks. The top tier processes the raw audio waveform at its original sampling rate, while lower tiers process downsampled versions of the input signal. The output of lower tiers is upsampled and combined with the input to the tier above it in order to provide a stronger conditioning signal. This architecture is inspired by related neural network architectures for AR modeling that incorporate multi-scale characteristics such as SampleRNN and PixelCNN++ [182].

The pooling is accomplished by simple reshaping and linear operations. Concretely, an input sequence $x \in \mathbb{R}^{T \times H}$ with context length T and hidden dimension size H is transformed through these shapes:

$$\begin{aligned} (\textbf{Down-pool}) (T, H) &\xrightarrow{\text{reshape}} (T/p, p \cdot H) \xrightarrow{\text{linear}} (T/p, q \cdot H) \\ (\textbf{Up-pool}) (T, H) &\xrightarrow{\text{linear}} (T, p \cdot H/q) \xrightarrow{\text{reshape}} (T \cdot p, H/q). \end{aligned}$$

Here, p is the *pooling factor* and q is an *expansion factor* that increases the hidden dimension while pooling. In our experiments, we always fix $p = 4, q = 2$ and use a total of just two pooling layers (three tiers).

We additionally note that in AR settings, the up-pooling layers must be shifted by a time step to ensure causality.

Table 8.3: Effect of context length on the performance of SASHIMI on Beethoven, controlling for sample and computation efficiency. SASHIMI is able to leverage information from longer contexts.

CONTEXT SIZE	BATCH SIZE	NLL	
		200K STEPS	10 HOURS
1 SECOND	8	1.364	1.433
2 SECONDS	4	1.229	1.298
4 SECONDS	2	1.120	1.234
8 SECONDS	1	1.007	1.095

8.4 Experiments

We evaluate SASHIMI on several benchmark audio generation and unconditional speech generation tasks in both AR and non-AR settings, validating that SASHIMI generates more globally coherent waveforms than baselines while having higher computational and sample efficiency.

Baselines. We compare SASHIMI to the leading AR models for unconditional waveform generation, SampleRNN and WaveNet. In Section 8.4.3, we show that SASHIMI can also improve non-AR models.

Datasets. We evaluate SASHIMI on datasets spanning music and speech generation (Table 8.1).

- **Beethoven.** A benchmark music dataset [135], consisting of Beethoven’s piano sonatas.
- **YouTubeMix.** Another piano music dataset [42] with higher-quality recordings than Beethoven.
- **SC09.** A benchmark speech dataset [49], consisting of 1-second recordings of the digits “zero” through “nine” spoken by many different speakers.

All datasets are quantized using 8-bit quantization, either linear or μ -law, depending on prior work. Each dataset is divided into non-overlapping chunks; the SampleRNN baseline is trained using TBPTT, while WaveNet and SASHIMI are trained on entire chunks. All models are trained to predict the negative log-likelihood (NLL) of individual audio samples; results are reported in base 2, also known as bits per byte (BPB) because of the one-byte-per-sample quantization. All datasets were sampled at a rate of 16kHz. Table 8.1 summarizes characteristics of the datasets and processing.

Table 8.4: Negative log-likelihoods and mean opinion scores on YouTubeMix. As suggested by Dieleman, Oord, and Simonyan [46], we encourage readers to form their own opinions by referring to the sound examples in our supplementary material.

MODEL	TEST NLL	MOS (FIDELITY)	MOS (MUSICALITY)
SAMPLERNN	1.723	2.98 ± 0.08	1.82 ± 0.08
WAVENET	1.449	2.91 ± 0.08	2.71 ± 0.08
SASHIMI	1.294	2.84 ± 0.09	3.11 ± 0.09
DATASET	-	3.76 ± 0.08	4.59 ± 0.07

8.4.1 Unbounded Music Generation

Because music audio is not constrained in length, AR models are a natural approach for music generation, since they can generate samples longer than the context windows they were trained on. We validate that SASHIMI can leverage longer contexts to perform music waveform generation more effectively than baseline AR methods.

We follow the setting of Mehri et al. [135] for the Beethoven dataset. Table 8.2 reports results found in prior work, as well as our reproductions. In fact, our WaveNet baseline is much stronger than the one implemented in prior work. SASHIMI substantially improves the test NLL by 0.09 BPB compared to the best baseline. Table 8.3 ablates the context length used in training, showing that SASHIMI significantly benefits from seeing longer contexts, and is able to effectively leverage extremely long contexts (over 100k steps) when predicting next samples.

Next, we evaluate all baselines on YouTubeMix. Table 8.4 shows that SASHIMI substantially outperforms SampleRNN and WaveNet on NLL. Following Dieleman, Oord, and Simonyan [46] (protocol in Appendix H.2.4), we measured mean opinion scores (MOS) for audio fidelity and musicality for 16s samples generated by each method (longer than the training context). All methods have similar fidelity, but SASHIMI substantially improves musicality by around 0.40 points, validating that it can generate long samples more coherently than other methods.

Figure 8.2 shows that SASHIMI trains stably and more efficiently than baselines in wall clock time. Appendix H.1, Figure H.2 also analyzes the peak throughput of different AR models as a function of batch size.

8.4.2 Model ablations: Slicing the SaShiMi

We validate some technical aspects of S4 and ablate SASHIMI’s architecture.

Figure 8.2: (**Sample Efficiency.**) Plot of validation NLL (in bits) vs. wall clock time (hours) on the SC09 dataset.

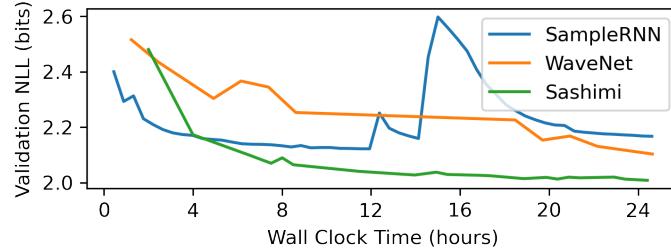


Table 8.5: Architectural ablations and efficiency tradeoffs on YouTubeMix. (*Top*) AR models and baselines at different sizes. (*Bottom*) Ablating the pooling layers of SASHIMI.

MODEL	NLL	TIME/EPOCH	THROUGHPUT	PARAMS
SAMPLERNN–2 TIER	1.762	800s	112K SAMPLES/S	51.85M
SAMPLERNN–3 TIER	1.723	850s	116K SAMPLES/S	35.03M
WAVENET–512	1.467	1000s	185K SAMPLES/S	2.67M
WAVENET–1024	1.449	1435s	182K SAMPLES/S	4.24M
SASHIMI–2 LAYERS	1.446	205s	596K SAMPLES/S	1.29M
SASHIMI–4 LAYERS	1.341	340s	316K SAMPLES/S	2.21M
SASHIMI–6 LAYERS	1.315	675s	218K SAMPLES/S	3.13M
SASHIMI–8 LAYERS	1.294	875s	129K SAMPLES/S	4.05M
ISOTROPIC S4–4 LAYERS	1.429	1900s	144K SAMPLES/S	2.83M
ISOTROPIC S4–8 LAYERS	1.524	3700s	72K SAMPLES/S	5.53M

Stabilizing S4

We consider how different parameterizations of S4’s representation affect downstream performance. Recall that S4 uses a DPLR state matrix $\mathbf{A} = \mathbf{\Lambda} + \mathbf{PQ}^*$ initialized to HIPPO, which theoretically captures long-range dependencies (Part II). We ablate various parameterizations of a small SASHIMI model (2 layers, 500 epochs on YouTubeMix). In particular, we consider the effect of the Hurwitz parameterization (Section 3.3.3), which uses the more constrained form $\mathbf{A} = \mathbf{\Lambda} - \mathbf{PP}^*$ and also adds constraints on $\mathbf{\Lambda}$ (Section 3.4.2). Without this Hurwitz parameterization, training \mathbf{A} yields consistent improvements over freezing \mathbf{A} , but becomes unstable at generation. The Hurwitz reparameterization allows \mathbf{A} to be learned while preserving stability, agreeing with the analysis in Section 3.3.3. A visual illustration of the spectral radii of the learned $\overline{\mathbf{A}}$ in the new parameterization is provided in Figure 8.3.

Table 8.6: (**SC09**) Automated metrics and human opinion scores. (*Top*) SASHIMI is the first AR model that can unconditionally generate high quality samples on this challenging dataset of fixed-length speech clips with highly variable characteristics. (*Middle*) As a flexible architecture for general waveform modeling, SASHIMI sets a true state-of-the-art when combined with a recent diffusion probabilistic model.

MODEL	PARAMS	NLL	FID ↓	IS ↑	MIS ↑	AM ↓	HUMAN (κ)	MOS		
							AGREEMENT	QUALITY	INTELLIGIBILITY	DIVERSITY
SAMPLERNNN	35.0M	2.042	8.96	1.71	3.02	1.76	0.321	1.18 ± 0.04	1.37 ± 0.02	2.26 ± 0.10
WAVENET	4.2M	1.925	5.08	2.27	5.80	1.47	0.408	1.59 ± 0.06	1.72 ± 0.03	2.70 ± 0.11
SASHIMI	4.1M	1.891	1.99	4.12	24.57	0.90	0.832	3.29 ± 0.07	3.53 ± 0.04	3.26 ± 0.09
SASHIMI*	5.8M	1.873	1.99	5.13	42.57	0.74	—	—	—	—
WAVEGAN	19.1M	-	2.03	4.90	36.10	0.80	0.840	2.98 ± 0.07	3.27 ± 0.04	3.25 ± 0.09
DIFFWAVE	24.1M	-	1.92	5.26	51.21	0.68	0.917	4.03 ± 0.06	4.15 ± 0.03	3.45 ± 0.09
W/ SASHIMI	23.0M	-	1.42	5.94	69.17	0.59	0.953	4.20 ± 0.06	4.33 ± 0.03	3.28 ± 0.11
TRAIN	-	-	0.00	8.56	292.5	0.16	0.921	4.04 ± 0.06	4.27 ± 0.03	3.59 ± 0.09
TEST	-	-	0.02	8.33	257.6	0.19	—	—	—	—

* AN UPDATED MODEL WITH A SYMMETRIC UNET STRUCTURE WAS RUN AFTER THE FIRST VERSION OF THESE EXPERIMENTS, WHICH ACHIEVES SUBSTANTIALLY BETTER AUTOMATED METRICS; MOS SCORES ARE NOT AVAILABLE.

TRAINED	FROZEN	NLL	STABLE GENERATION
—	$\Lambda + PQ^*$	1.445	✓
$\Lambda + PQ^*$	—	1.420	✗
$\Lambda - PP^*$	—	1.419	✓

Multi-scale architecture. We investigate the effect of SASHIMI’s architecture (Section 8.3) against isotropic S4 layers on YouTubeMix. Controlling for parameter counts, adding pooling in SASHIMI leads to substantial improvements in computation and modeling (Table 8.5, Bottom).

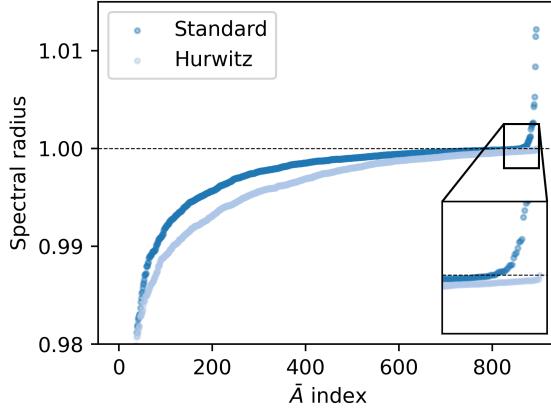
Efficiency tradeoffs. We ablate different sizes of the SASHIMI model on YouTubeMix to show its performance tradeoffs along different axes.

Table 8.5 (Top) shows that a single SASHIMI model simultaneously outperforms all baselines on quality (NLL) and computation at both training and inference, with a model more than 3× smaller. We note that for WaveNet, we use a fast, cached implementation for generation [151]. Moreover, SASHIMI improves monotonically with depth, suggesting that quality can be further improved at the cost of additional computation.

8.4.3 Unconditional Speech Generation

The SC09 spoken digits dataset is a challenging unconditional speech generation benchmark, as it contains several axes of variation (words, speakers, microphones, alignments).

Figure 8.3: **(S4 Stability)** Comparison of spectral radii for all \bar{A} matrices in a SaShiMi model trained with different S4 parameterizations. The instability in the standard S4 parameterization is solved by our Hurwitz parameterization.



Unlike the music setting (Section 8.4.1), SC09 contains audio of *bounded* length (1-second utterances). To date, AR waveform models trained on this benchmark have yet to generate spoken digits which are consistently intelligible to humans.¹ In contrast, non-AR approaches are capable of achieving global coherence on this dataset, as first demonstrated by WaveGAN [49].

Although our primary focus thus far has been the challenging testbed of AR waveform modeling, SASHIMI can also be used as a flexible neural network architecture for audio generation more broadly. We demonstrate this potential by integrating SASHIMI into DiffWave [112], a diffusion-based method for non-AR waveform generation which represents the current state-of-the-art for SC09. DiffWave uses the original WaveNet architecture as its backbone—here, we simply replace WaveNet with a SASHIMI model containing a similar number of parameters.

We compare SASHIMI to strong baselines on SC09 in both the AR and non-AR (via DiffWave) settings by measuring several standard quantitative and qualitative metrics such as Frechét Inception Distance (FID) and Inception Score (IS) (Appendix H.2.3). We also conduct a qualitative evaluation where we ask several annotators to label the generated digits and then compute their inter-annotator agreement. Additionally, as in Donahue, McAuley, and Puckette [49], we ask annotators for their subjective opinions on overall audio quality, intelligibility, and speaker diversity, and report MOS for each axis. Results for all models

¹While AR waveform models can produce intelligible speech in the context of TTS systems, this capability requires conditioning on rich intermediaries like spectrograms or linguistic features.

Table 8.7: (**SC09 diffusion models.**) Beyond AR, SASHIMI can be flexibly combined with other generative modeling approaches, improving on the state-of-the-art DiffWave model by simply replacing the architecture. (*Top*) A parameter-matched SASHIMI architecture with *no tuning* outperforms the best DiffWave model. (*Middle*) SASHIMI is consistently better than WaveNet at all stages of training; a model trained on half the samples matches the best DiffWave model. (*Bottom*) The WaveNet backbone is extremely sensitive to architecture parameters such as size and dilation schedule; a small model fails to learn. We also ablate the bi-directional S4 layer, which outperforms the uni-directional one with the same training speed.

ARCHITECTURE	PARAMS	TRAINING STEPS	FID ↓	IS ↑	MIS ↑	AM ↓	NDB ↓	HUMAN (κ) AGREEMENT	MOS		
									QUALITY	INTELLIGIBILITY	DIVERSITY
SASHIMI WAVENET	23.0M	800K	1.42	5.94	69.17	0.59	0.88	0.953	4.20 ± 0.06	4.33 ± 0.03	3.28 ± 0.11
	24.1M	1000K	1.92	5.26	51.21	0.68	0.88	0.917	4.03 ± 0.06	4.15 ± 0.03	3.45 ± 0.09
SASHIMI WAVENET	23.0M	500K	2.08	5.68	51.10	0.66	0.76	0.923	3.99 ± 0.06	4.13 ± 0.03	3.38 ± 0.10
	24.1M	500K	2.25	4.68	34.55	0.80	0.90	0.848	3.53 ± 0.07	3.69 ± 0.03	3.30 ± 0.08
SASHIMI (UNI.)	7.1M	500K	2.70	3.62	17.96	1.03	0.90	0.829	3.08 ± 0.07	3.29 ± 0.04	3.26 ± 0.08
SASHIMI WAVENET	7.5M	500K	1.70	5.00	40.27	0.72	0.90	0.934	3.83 ± 0.07	4.00 ± 0.03	3.34 ± 0.09
	6.8M	500K	4.53	2.80	9.02	1.30	0.94	0.446	1.85 ± 0.08	1.90 ± 0.03	3.03 ± 0.10

appear in Table 8.6.

Autoregressive. SASHIMI substantially outperforms other AR waveform models on all metrics, and achieves $2\times$ higher MOS for both quality and intelligibility. Moreover, annotators agree on labels for samples from SASHIMI far more often than they do for samples from other AR models, suggesting that SASHIMI generates waveforms that are more globally coherent on average than prior work. Finally, SASHIMI achieves higher MOS on all axes compared to WaveGAN while using more than $4\times$ fewer parameters.

Non-autoregressive. Integrating SASHIMI into DiffWave substantially improves performance on all metrics compared to its WaveNet-based counterpart, and achieves a new overall state-of-the-art performance on all quantitative and qualitative metrics on SC09. We note that this result involved *zero tuning* of the model or training parameters (e.g. diffusion steps or optimizer hyperparameters) (Appendix H.2.2). This suggests that SASHIMI could be useful not only for AR waveform modeling but also as a new drop-in architecture for many audio generation systems which currently depend on WaveNet (see Section 8.2).

We additionally conduct several ablation studies on our hybrid DiffWave and SASHIMI model, and compare performance earlier in training and with smaller models (Table 8.7). When paired with DiffWave, SASHIMI is much more sample efficient than WaveNet, matching the performance of the best WaveNet-based model with half as many training steps. Kong et al. [112] also observed that DiffWave was extremely sensitive with a WaveNet backbone, performing poorly with smaller models and becoming unstable with larger ones. We

show that, when using WaveNet, a small DiffWave model fails to model the dataset, however it works much more effectively when using SASHIMI. Finally, we ablate the non-causal relaxation of the S4 architecture in Section 7.2.6, showing that this bidirectional version of SASHIMI performs much better than its unidirectional counterpart (as expected for this task that does not require causality).

8.5 Discussion

Our results indicate that SASHIMI is a promising new architecture for modeling raw audio waveforms. When trained on music and speech datasets, SASHIMI generates waveforms that humans judge to be more musical and intelligible respectively compared to waveforms from previous architectures, indicating that audio generated by SASHIMI has a higher degree of global coherence. By leveraging the dual convolutional and recurrent forms of S4, SASHIMI is more computationally efficient than past architectures during both training and inference. Additionally, SASHIMI is consistently more sample efficient to train—it achieves better quantitative performance with fewer training steps. Finally, when used as a drop-in replacement for WaveNet, SASHIMI improved the performance of an existing state-of-the-art model for unconditional generation, indicating a potential for SASHIMI to create a ripple effect of improving audio generation more broadly.

Chapter 9

S4ND: Extending S4 to Multi-dimensional Signals

All methods and results so far have focused on 1-dimensional sequences. In this section, we show that S4 can be generalized to higher-dimensional signals and sequences, which can be naturally applied to visual data such as images and videos, which can be viewed as discretizations of continuous multi-dimensional signals. We show that S4ND can model large-scale visual data in 1-D, 2-D, and 3-D with strong performance by simply swapping out for other primitives such as Conv2D or self-attention layers, while acquiring additional abilities such as zero-shot resolution adaptation.

9.1 Introduction

Modeling visual data such as images and videos is a canonical problem in deep learning. In the last few years, many modern deep learning backbones that achieve strong performance on benchmarks like ImageNet [179] have been proposed. These backbones are diverse, and include 1D sequence models such as the Vision Transformer (ViT) [51], which treats images as sequences of patches, and 2D and 3D models that use local convolutions over images and videos (ConvNets) [115, 81, 191, 195, 200, 131, 78, 97, 167, 210, 59].

A commonality among modern vision models capable of achieving state-of-the-art (SOTA) performance is that they treat visual data as discrete pixels rather than continuous-signals. However, images and videos are discretizations of multidimensional and naturally continuous signals, sampled at a fixed rate in the spatial and temporal dimensions. Ideally, we would

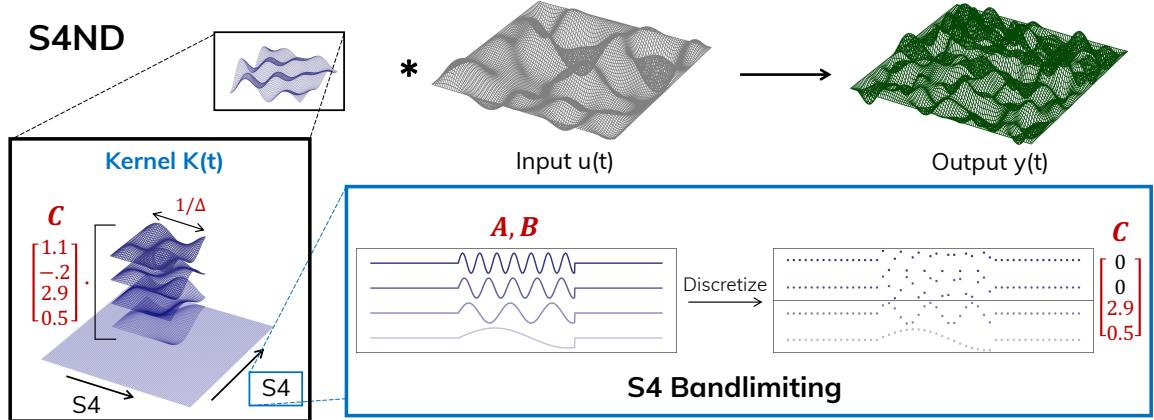


Figure 9.1: (S4ND.) (Top) S4ND can be viewed as a depthwise convolution that maps a multidimensional input (black) to output (green) through a continuous convolution kernel (blue). (Bottom Left) The kernel can be interpreted as a linear combination (controlled by \mathbf{C}) of basis functions (controlled by \mathbf{A}, \mathbf{B}) with flexible width (controlled by step size Δ). For structured \mathbf{C} , the kernel can further factorize as a low-rank tensor product of 1D kernels, and can be interpreted as independent S4 transformations on each dimension. (Bottom Right) Choosing \mathbf{A}, \mathbf{B} appropriately yields Fourier basis functions with controllable frequencies. To avoid aliasing in the final discrete kernels, the coefficients of \mathbf{C} corresponding to high frequencies can simply be masked out.

want approaches that are capable of recognizing this distinction between data and signal, and directly model the underlying continuous-signals. This would give them capabilities like the ability to adapt the model to data sampled at different resolutions.

A natural approach to building such models is to parameterize and learn continuous convolutional kernels, which can then be sampled differently for data at different resolutions [60, 173, 184, 74, 71]. Among these, S4 models have achieved SOTA results in modeling sequence data derived from continuous-signals, such as audio (Chapters 7 and 8). However, a key limitation of S4 is that it was developed for 1D signals, and cannot directly be applied to visual data derived from multidimensional “ND” signals. Given that 1D SSMs outperform other continuous modeling solutions for sequence data [71], and have had preliminary success on image [71] and video classification [93], we hypothesize that they may be well suited to modeling visual data when appropriately generalized to the setting of multidimensional signals.

This chapter’s main contribution is S4ND, an extension of S4 to multidimensional signals. The key idea is to turn the standard SSM (a 1D ODE) into a multidimensional PDE governed by an independent SSM per dimension. By adding additional structure to this ND

SSM, we show that it is equivalent to an ND continuous convolution that can be factored into a separate 1D SSM convolution per dimension. This results in a model that is efficient and easy to implement, using the standard 1D S4 layer as a black box. Furthermore, it can be controlled by S4’s parameterization, allowing it to model both long-range dependencies, or finite windows with a learnable window size that generalize conventional local convolutions (Chapter 5).

We show that S4ND can be used as a drop-in replacement in strong modern vision architectures while matching or improving performance in 1D, 2D, and 3D. With minimal change to the training procedure, replacing the self-attention in ViT with S4-1D improves top-1 accuracy by 1.5%, and replacing the convolution layers in a 2D ConvNeXt backbone [131] with S4-2D preserves its performance on ImageNet-1k [43]. Simply inflating (temporally) this pretrained S4-2D-ConvNeXt backbone to 3D improves video activity classification results on HMDB-51 [116] by 4 points over the pretrained 3D ConvNeXt baseline. Notably, we use S4ND as global kernels that span the entire input shape, which enable it to have global context (both spatially and temporally) in every layer of a network.

Additionally, we propose a low-pass bandlimiting modification to S4 that encourages the learned convolutional kernels to be smooth. While S4ND can be used at any resolution, performance suffers when moving between resolutions due to aliasing artifacts in the kernel, an issue also noted by prior work on continuous models [173]. While S4 was capable of transferring between different resolutions on audio data (Section 7.4), visual data presents a greater challenge due to the scale-invariant properties of images in space and time [177], as sampled images with more distant objects are more likely to contain power at frequencies above the Nyquist cutoff frequency. Motivated by this, we propose a simple criteria that masks out frequencies in the S4ND kernel that lie above the Nyquist cutoff frequency.

The continuous-signal modeling capabilities of S4ND open the door to new training recipes, such as the ability to train and test at different resolutions. On the standard CIFAR-10 [114] and Celeb-A [132] datasets, S4ND degrades by as little as 1.3% when upsampling from low- to high-resolution data (e.g. $128 \times 128 \rightarrow 160 \times 160$), and can be used to facilitate progressive resizing to speed up training by 22% with $\sim 1\%$ drop in final accuracy compared to training at the high resolution alone. We also validate that our new bandlimiting method is critical to these capabilities, with ablations showing absolute performance degradation of up to 20%+ without it.

9.2 Related Work

Image Classification. There is a long line of work in image classification, with much of the 2010s dominated by ConvNet backbones [115, 81, 191, 195, 200]. Recently, Transformer backbones, such as ViT [51], have achieved SOTA performance on images using self-attention over a sequence of 1D patches [129, 128, 207, 235, 48]. Their scaling behavior in both model and dataset training size is believed to give them an inherent advantage over ConvNets [51], even with minimal inductive bias. Liu et al. [131] introduce ConvNeXt, which modernizes the standard ResNet architecture [81] using modern training techniques, matching the performance of Transformers on image classification. We select a backbone in the 1D and 2D settings, ViT and ConvNeXt, to convert into continuous-signal models by replacing the multi-headed self-attention layers in ViT and the standard Conv2D layers in ConvNeXt with S4ND layers, boosting or maintaining their top-1 accuracy on large-scale image classification.

S4 & Video Classification. To handle the long-range dependencies inherent in videos, [93] used 1D S4 for video classification on the Long-form Video Understanding dataset [226]. They first applied a Transformer to each frame to obtain a sequence of patch embeddings for each video frame independently, followed by a standard 1D S4 to model across the concatenated sequence of patches. This is akin to previous methods that learned spatial and temporal information separately [103], for example using ConvNets on single frames, followed by an LSTM [87] to aggregate temporal information. In contrast, modern video architectures such as 3D ConvNets and Transformers [78, 97, 167, 210, 59, 111, 130, 226, 6, 2] show stronger results when learning spatiotemporal features simultaneously, which the generalization of S4ND into multidimensions now enables us to do.

Continuous-signal Models. Visual data are discretizations of naturally continuous signals that possess extensive structure in the joint distribution of spatial frequencies, including the properties of scale and translation invariance. For example, an object in an image generates correlations between lower and higher frequencies that arises in part from phase alignment at edges [145]. As an object changes distances in the image, these correlations remain the same but the frequencies shift. This relationship can potentially be learned from a coarsely sampled image and then applied at higher frequency at higher resolution.

A number of continuous-signal models have been proposed for the visual domain to learn these inductive biases, and have led to additional desirable properties and capabilities. A classic example of continuous-signal driven processing is the fast Fourier transform, which

is routinely used for filtering and data consistency in computational and medical imaging [44]. Neural Radiance Fields (NeRF) represents a static scene as a continuous function, allowing them to render scenes smoothly from multiple viewpoints [138]. CKConv [174] learns a continuous representation to create kernels of arbitrary size for several data types including images, with additional benefits such as the ability to handle irregularly sampled data. FlexConv [173] extends this work with a learned kernel size, and show that images can be trained at low resolution and tested at high resolution if the aliasing problem is addressed. S4 [71] increased abilities to model long-range dependencies using continuous kernels, allowing SSMs to achieve SOTA on sequential CIFAR [114]. However, these methods including 1D S4 have been applied to relatively low dimensional data, e.g., time series, and small image datasets. S4ND is the first continuous-signal model applied to high dimensional visual data with the ability to maintain SOTA performance on large-scale image and video classification.

Progressive Resizing. Training times for large-scale image classification can be quite long, a trend that is exacerbated by the emergence of foundation models [16]. A number of strategies have emerged for reducing overall training time. Fix-Res [209] trains entirely at a lower resolution, and then fine-tunes at the higher test resolution to speed up training in a two-stage process. Mix-and-Match [89] randomly samples low and high resolutions during training in an interleaved manner. An effective method to reduce training time on images is to utilize progressive resizing. This involves training at a lower resolution and gradually upsampling in stages. For instance, *Training imagenet in 3 hours for 25 minutes* [57] utilized progressive resizing to train an ImageNet in under 4 hours. EfficientNetV2 [200] coupled resizing with a progressively regularization schedule, increasing the regularization as well to maintain accuracy. In EfficientNetV2 and other described approaches, the models eventually train on the final test resolution. As a continuous-signal model, we demonstrate that S4ND is naturally suited to progressive resizing, while being able to generalize to *unseen* resolutions at test time.

9.3 Method

We describe the proposed S4ND model for the 2D case only, for ease of notation and presentation. The results extend readily to general dimensions; full statements and proofs for the general case are in Appendix I.1. Section 9.3.1 describes the multidimensional S4ND layer, and Section 9.3.2 describes our simple modification to restrict frequencies in the kernels. Figure 9.1 illustrates the complete S4ND layer.

9.3.1 S4ND

We begin by generalizing the (linear time-invariant) SSM (2.1) to higher dimensions. Notationally, we denote the individual time axes with superscripts in parentheses. Let $u = u(t^{(1)}, t^{(2)})$ and $y = y(t^{(1)}, t^{(2)})$ be the input and output which are signals $\mathbb{R}^2 \rightarrow \mathbb{C}$, and $x = (x^{(1)}(t^{(1)}, t^{(2)}), x^{(2)}(t^{(1)}, t^{(2)})) \in \mathbb{C}^{N^{(1)} \times N^{(2)}}$ be the SSM state of dimension $N^{(1)} \times N^{(2)}$, where $x^{(\tau)} : \mathbb{R}^2 \rightarrow \mathbb{C}^{N^{(\tau)}}$.

Definition 9.1 (Multidimensional SSM). *Given parameters $\mathbf{A}^{(\tau)} \in \mathbb{C}^{N^{(\tau)} \times N^{(\tau)}}$, $\mathbf{B}^{(\tau)} \in \mathbb{C}^{N^{(\tau)} \times 1}$, $\mathbf{C} \in \mathbb{C}^{N^{(1)} \times N^{(2)}}$, the 2D SSM is the map $u \mapsto y$ defined by the linear PDE with initial condition $x(0, 0) = 0$:*

$$\begin{aligned}\frac{\partial}{\partial t^{(1)}} x(t^{(1)}, t^{(2)}) &= (\mathbf{A}^{(1)} x^{(1)}(t^{(1)}, t^{(2)}), x^{(2)}(t^{(1)}, t^{(2)})) + \mathbf{B}^{(1)} u(t^{(1)}, t^{(2)}) \\ \frac{\partial}{\partial t^{(2)}} x(t^{(1)}, t^{(2)}) &= (x^{(1)}(t^{(1)}, t^{(2)}), \mathbf{A}^{(2)} x^{(2)}(t^{(1)}, t^{(2)})) + \mathbf{B}^{(2)} u(t^{(1)}, t^{(2)}) \\ y(t^{(1)}, t^{(2)}) &= \langle \mathbf{C}, x(t^{(1)}, t^{(2)}) \rangle\end{aligned}\quad (9.1)$$

Note that Definition 9.1 differs from the usual notion of multidimensional SSM, which is simply a map from $u(t) \in \mathbb{C}^n \mapsto y(t) \in \mathbb{C}^m$ for higher-dimensional $n, m > 1$ but still with 1 time axis. However, Definition 9.1 is a map from $u(t_1, t_2) \in \mathbb{C}^1 \mapsto y(t_1, t_2) \in \mathbb{C}^1$ for *scalar* input/outputs but over *multiple* time axes. When thinking of the input $u(t^{(1)}, t^{(2)})$ as a function over a 2D grid, Definition 9.1 can be thought of as a simple linear PDE that just runs a standard 1D SSM over each axis independently.

Analogous to equation (2.2), the 2D SSM can also be viewed as a multidimensional convolution.

Theorem 9.2. (9.1) is a time-invariant system that is equivalent to a 2D convolution $y = K * u$ by the kernel

$$K(t^{(1)}, t^{(2)}) = \langle \mathbf{C}, (e^{t^{(1)} \mathbf{A}^{(1)}} \mathbf{B}^{(1)}) \otimes (e^{t^{(2)} \mathbf{A}^{(2)}} \mathbf{B}^{(2)}) \rangle \quad (9.2)$$

This kernel is a linear combination of the $N^{(1)} \times N^{(2)}$ basis kernels $\{K_{n^{(1)}}^{(1)}(t^{(1)}) \otimes K_{n^{(2)}}^{(2)}(t^{(2)}) : n^{(1)} \in [N^{(1)}], n^{(2)} \in [N^{(2)}]\}$ where $K^{(\tau)}$ are the standard 1D SSM kernels (2.2) for each axis

However, a limitation of this general form is that the number of basis functions $N^{(1)} \times N^{(2)} \times \dots$ grows exponentially in the dimension, increasing the parameter count (of \mathbf{C})

and overall computation dramatically. This can be mitigated by factoring \mathbf{C} as a low-rank tensor.

Corollary 9.3. *Suppose that $\mathbf{C} \in \mathbb{C}^{N^{(1)} \times N^{(2)}}$ is a low-rank tensor $\mathbf{C} = \sum_{i=1}^r \mathbf{C}_i^{(1)} \otimes \mathbf{C}_i^{(2)}$ where each $\mathbf{C}_i^{(\tau)} \in \mathbb{C}^{N^{(\tau)}}$. Then the kernel (9.2) also factors as a tensor product of 1D kernels*

$$K(t^{(1)}, t^{(2)}) = \sum_{i=1}^r K_i^{(1)}(t^{(1)}) \otimes K_i^{(2)}(t^{(2)}) := \sum_{i=1}^r (\mathbf{C}_i^{(1)} e^{t^{(2)} \mathbf{A}^{(1)}} \mathbf{B}^{(1)}) \otimes (\mathbf{C}_i^{(2)} e^{t^{(2)} \mathbf{A}^{(2)}} \mathbf{B}^{(2)})$$

In our experiments, we choose \mathbf{C} as a rank-1 tensor, but the rank can be freely adjusted to tradeoff parameters and computation for expressivity. Using the equivalence between (2.1) and (2.2), Corollary 9.3 also has the simple interpretation as defining an independent 1D SSM along each axis of the multidimensional input.

9.3.2 Resolution Change and Bandlimiting

SSMs in 1D have shown strong performance in the audio domain, and can nearly preserve full accuracy when tested zero-shot on inputs sampled at very different frequencies (Section 7.4.1). This capability relies simply on scaling Δ by the relative change in frequencies (i.e., if the input resolution is doubled, halve the SSM’s Δ parameter). However, sampling rates in the spatial domain are often much lower than temporally, leading to potential aliasing when changing resolutions. A standard technique to avoid aliasing is to apply a low-pass filter to remove frequencies above the Nyquist cutoff frequency.

For example, when \mathbf{A} is diagonal with n -th element \mathbf{A}_n , each basis function has simple form $K_n(t) = e^{t\mathbf{A}_n} \mathbf{B}_n$. (see Definition 2.18 and Remark 6.3). Note that the frequencies are mainly controlled by the imaginary part of \mathbf{A}_n . We propose the following simple method: for any n such that $\mathbf{A}_n \cdot \Delta < \frac{1}{2}\alpha$, mask out the corresponding coefficient of the linear combination \mathbf{C}_n . Here α is a hyperparameter that controls the cutoff; theoretically, $\alpha = 1.0$ corresponds to the Nyquist cutoff if the basis functions are pure sinusoids. However, due to the decay $e^{\Re(\mathbf{A}_n)}$ arising from the real part as well as approximations arising from using finite-state SSMs, α often has to be set lower empirically.

9.4 Experiments

We evaluate S4ND on large-scale image classification in Section 9.4.1 in the 1D and 2D settings, followed by activity classification in videos in Section 9.4.2 in the 3D setting, where using S4ND as a drop-in replacement for standard deep learning layers matches or improves performance in all settings. In Section 9.4.3, we performed controlled ablations to highlight the benefits of S4ND as a continuous-signal model in images.

9.4.1 S4ND in 1D & 2D: Large-scale Image Classification

First, we show that S4ND is a drop-in replacement for existing visual modeling layers such as 1D self-attention and 2D local convolutions, with no degradation in top-1 performance when used in modern backbones such as ViT [51] and ConvNeXt [131] on ImageNet-1k [43].

Baselines and Methodology. We consider large-scale image classification on the ImageNet-1k dataset, which consists of 1000 classes and 1.3M images. We start with two strong baselines: ViT-B (base, 88M parameters) for processing images in the 1D setting and ConvNeXt-T (tiny, 28.4M) in the 2D setting. (We omit the postfix “B” and “T” for brevity). More recent works using Transformers on images have surpassed ViT, but we focus on the original ViT model to highlight specifically the drop-in capability and performance difference in self-attention vs. S4ND layers. We first swap the self-attention layers in ViT with S4ND layers, and call this model S4ND-ViT. Notably, we simplify ViT by removing the positional encodings, as S4ND does not require injecting this inductive bias. Similarly, we swap the local Conv2D layers in the ConvNeXt blocks with S4ND layers, which we call S4ND-ConvNeXt, a model with global context at each layer.

Table 9.1: (**Performance on image classification.**) Top-1 test accuracy benchmarks for images in the 1D and 2D settings. ConvNeXt-M, for “micro”, is a reduced model size for Celeb-A, while “-ISO” is an isotropic S4ND backbone using the architecture in Section 7.2.

MODEL	DATASET	PARAMS	ACC
ViT-B	ImageNet	88.0M	78.9
S4ND-ViT-B	ImageNet	88.8M	80.4
ConvNeXt-T	ImageNet	28.4M	82.1
S4ND-ConvNeXt-T	ImageNet	30.0M	82.2
Conv2D-ISO	CIFAR-10	2.2M	93.7
S4ND-ISO	CIFAR-10	5.3M	94.1
ConvNeXt-M	Celeb-A	9.2M	91.0
S4ND-ConvNeXt-M	Celeb-A	9.6M	91.3

Both S4ND variants result in similar parameter counts compared to their baseline models.

Training. For all ImageNet models, we train from scratch with no outside data and adopt the training procedure from [207, 231], which uses the AdamW optimizer [133] for 300 epochs, cosine decay learning rate, weight decay 0.05, and aggressive data augmentations including RandAugment [34], Mixup [237], and AugMix [83]. We add RepeatAug [88] for ConvNeXt and S4ND-ConvNeXt. The initial learning rate for ViT (and S4ND-ViT) is 0.001, while for ConvNeXt (and S4ND-ConvNeXt) it is 0.004. See Appendix I.2.1 for additional training procedure details.

Results. Table 9.1 shows top-1 accuracy results for each model on ImageNet. After reproducing the baselines, S4ND-ViT was able to moderately boost performance by +1.5% over ViT, while S4ND-ConvNeXt matched the original ConvNeXt’s performance. This indicates that S4ND is a strong primitive that can replace self-attention and standard 2D convolutions in practical image settings with large-scale data.

9.4.2 S4ND in 3D: Video Classification

Next, we demonstrate the flexible capabilities of S4ND in settings involving pretraining and even higher-dimensional signals. We use the activity recognition dataset HMDB-51 [116] which involves classifying videos in 51 activity classes.

Baselines and Methodology. Prior work demonstrated that 2D CNNs (e.g. pretrained on ImageNet) can be adapted to 3D models by 2D to 3D kernel *inflation* (I3D [20]), in which the 2D kernels are repeated temporally N times and normalized by $1/N$. Our baseline, which we call ConvNeXt-I3D, uses the 2D ConvNeXt pretrained on ImageNet (Section 9.4.1) with I3D inflation. Notably, utilizing S4ND in 3D enables global context *temporally* as well. We additionally test the more modern spatial-temporal separated 3D convolution used by S3D [229] and R(2+1)D [211], which factor the 3D convolution kernel as the outer product of a 2D (spatial) by 1D (temporal) kernel. Because of its flexible factored form (Section 9.3.1), S4ND automatically has these inflation capabilities. We inflate the pretrained S4ND-ConvNeXt simply by loading the pretrained 2D model weights for the spatial dimensions, and initializing the temporal kernel parameters $\mathbf{A}^{(3)}, \mathbf{B}^{(3)}, \mathbf{C}^{(3)}$ from scratch. We note that this model is essentially identical to the baseline ConvNeXt-S3D except that each component of the factored kernels use standard 1D S4 layers instead of

1D local convolutions. Finally, by varying the initialization of these parameters, we can investigate additional factors affecting model training; in particular, we also run an ablation on the kernel timescales Δ .

Training. Our training procedure is minimal, using only RGB frames (no optical flow). We sample clips of 2 seconds with 30 total frames at 224×224 , followed by applying RandAugment; we performed a small sweep of the RandAugment magnitude for each model. All models are trained with learning rate 0.0001 and weight decay 0.2. Additional details are included in Appendix I.2.2.

Table 9.2: (**HMDB-51 Activity Recognition with ImageNet-pretrained models.**) (*Left*) Top-1 accuracy with 2D to 3SD kernel inflation. (*Right*) Ablation of initial temporal kernel lengths, controlled by S4’s Δ parameter.

	PARAMS	FLOW	RGB	INIT. LENGTH	ACC
Inception-I3D	25.0M	61.9	49.8	20.0	53.74
ConvNeXt-I3D	28.5M	-	58.1	4.0	58.33
ConvNeXt-S3D	27.9M	-	58.6	2.0	60.30
S4ND-ConvNeXt-3D	31.4M	-	62.1	1.0	62.07

Results. Results are presented in Table 9.2. Our baselines are much stronger than prior work in this setting, 8% top-1 accuracy higher than the original I3D model in the RGB frames only setting, and confirming that separable kernels (S3D) perform at least as well as standard inflation (+0.53%). S4ND-ConvNeXt-3D improves over the baseline ConvNeXt-I3D by +4.0% with no difference in models other than using a temporal S4 kernel. This even exceeds the performance of I3D when trained on optical flow.

Finally, we show how S4ND’s parameters can control for factors such as the kernel length (Table 9.2). Note that our temporal kernels $K^3(t^3)$ are always full length (30 frames in this case), while standard convolution kernels are shorter temporally and require setting the width of each layer manually as a hyperparameter [229]. S4 layers have a parameter Δ that can be interpreted such that $\frac{1}{\Delta}$ is the expected length of the kernel (Section 5.4). By simply adjusting this hyperparameter, S4ND can be essentially initialized with length-1 temporal kernels that can automatically learn to cover the whole temporal length if needed. We hypothesize that this contributes to S4ND’s improved performance over baselines.

Table 9.3: (**Settings for continuous capabilities experiments.**) Datasets and resolutions used for continuous capabilities experiments, as well as the model backbones used are summarized.

DATASET	CLASSES	RESOLUTION			BACKBONE
		base	mid	low	
CIFAR-10	10	32×32	$16 \times 16 (2\times)$	$8 \times 8 (4\times)$	Isotropic
Celeb-A	40 multilabel	160×160	$128 \times 128 (1.25\times)$	$64 \times 64 (2.50\times)$	ConvNeXt

Table 9.4: (**Zero-shot resolution change.**) Results for models trained on one resolution (one of low / mid / base), and zero-shot tested on another. Results are averaged over 2 random seeds.

RESOLUTION		CIFAR-10			CELEB-A	
TRAIN	TEST	S4ND	CONV2D	FLEXNET-16	S4ND	CONV2D
base	base	93.10 ± 0.22	91.9 ± 0.2	92.2 ± 0.1	91.75 ± 0.00	91.44 ± 0.03
mid	mid	88.80 ± 0.12	87.2 ± 0.1	86.5 ± 2.0	91.63 ± 0.04	91.09 ± 0.08
mid	base	88.77 ± 0.03	73.1 ± 0.3	82.7 ± 2.0	90.14 ± 0.38	80.52 ± 0.08
low	low	78.17 ± 0.13	76.0 ± 0.2	-	90.95 ± 0.02	90.37 ± 0.04
low	mid	78.86 ± 0.22	57.4 ± 0.3	-	84.44 ± 1.04	80.45 ± 0.11
low	base	73.71 ± 0.47	33.1 ± 1.3	-	84.73 ± 0.54	80.59 ± 0.14

CIFAR-10: Zero-Shot Test Resolution Performance

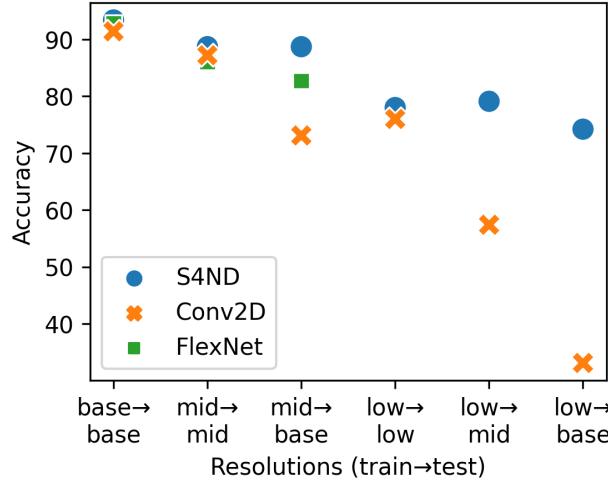


Figure 9.2: (**CIFAR-10 zero-shot comparison.**) When trained and tested on the same resolutions, all models have similar performance (with S4ND slightly better). But, when trained and tested on different resolutions (the zero-shot setting), S4ND significantly outperforms Conv2D and FlexNet.

9.4.3 Continuous-signal Capabilities for Images

Images are often collected at varied resolutions, even with the same hardware, so it is desirable that models generalize to data sampled at different resolutions. We show that S4ND inherits this capability as a continuous-signal model, with strong zero-shot performance when changing resolutions, and the ability to train with progressively resized multi-resolution data. We perform an ablation to show that our proposed bandlimiting modification is critical to achieving strong performance when changing resolutions.

Setup. We focus on image classification on 2D benchmark datasets: a dataset with low-resolution images (CIFAR-10) and one with higher-resolution images (Celeb-A). For each dataset, we specify a base image resolution (base), and two lower resolutions (mid, low), summarized in Table 9.3. To highlight that S4ND’s continuous capabilities are independent of backbone, we experiment with 2 different 2D backbones: an isotropic, fixed-width model backbone on CIFAR-10 [114] and a small ConvNeXt backbone on Celeb-A [132]. For each backbone, we compare S4ND’s performance to Conv2D layers as a standard, widely used baseline. Additional details can be found in Appendix I.2.3.

We first verify that S4ND achieves comparable test classification performance with baseline Conv2D models. Table 9.4 and Figure 9.2 show that we exceed the performance of Conv2D on both tasks, with S4ND improving over Conv2D models by 0.4% on CIFAR-10 and 0.3% on Celeb-A.

Zero-Shot Resolution Change. We train S4ND and Conv2D models at either low or mid resolution, and test them at the base resolution for each dataset. We also compare to FlexConv [173] on CIFAR-10, which is the current SOTA for zero-shot resolution change. Compared to training and testing at the base resolution, we expect that Conv2D should degrade more strongly than S4ND, since it cannot adapt its kernel appropriately to the changed resolution. Table 9.4 and Figure 9.2 show that S4ND outperforms Conv2D on mid → base by 15+ points and low → base by 40+ points on CIFAR-10, and 9+ points and 3+ points on Celeb-A. In fact, S4ND yields better performance on low → base (a more difficult task) than Conv2D does from mid → base (an easier task), improving by 1.1% on CIFAR-10 and 3% on Celeb-A. Compared to FlexConv on CIFAR-10 mid → base, S4ND improves zero-shot performance by 5+ points, setting a new SOTA.

Progressively Resized Training. We provide an exploration of training with progressive resizing [57, 200] i.e. training in multiple stages at different resolutions. The only change we make from standard training is to reset the learning rate scheduler at the beginning of

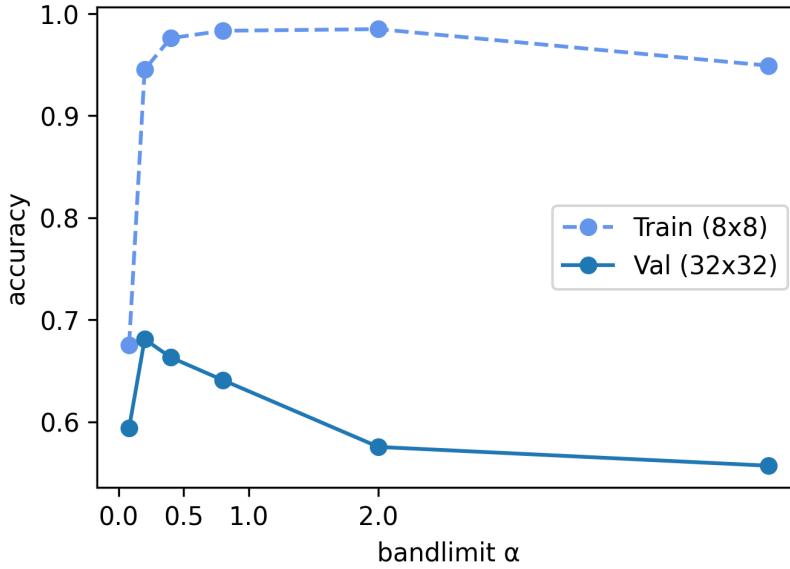


Figure 9.3: **(Bandlimiting ablation.)** Zero-shot performance training at 8×8 and evaluating at 32×32 . Train performance is high for large enough values of $\alpha \geq 0.5$, but validation performance goes down as aliasing occurs in the kernel. Both train and validation drop for lower values of α , as it limits the expressivity of the kernel.

each stage (details in Appendix I.2.3). We compare S4ND and the Conv2D baseline with progressive resizing in Table 9.5.

For CIFAR-10, we train with a low \rightarrow base, 80 – 20 epoch schedule, and perform within $\sim 1\%$ of an S4ND model trained with base resolution data while speeding up training by 21.8%. We note that Conv2D attains much higher speedups as a consequence of highly optimized implementations (see related discussion in Section 10.3). For Celeb-A, we explore flexibly combining the benefits of both progressive resizing and zero-shot testing, training with a low \rightarrow mid, 16 – 4 epoch schedule that uses no base data. We outperform Conv2D by 7.5%+, and attain large speedups of 50%+ over training at the base resolution.

Table 9.5: **(Progressive resizing results.)** Validation performance for progressively resized training at base resolution, and speedup compared to training at base resolution on CIFAR-10 and Celeb-A. We use a 80 – 20 and 16 – 4 schedule for CIFAR-10 and Celeb-A, and also report performance training only at base resolution.

DATASET	MODEL	EPOCH SCHEDULE	TRAIN RESOLUTION	VAL @ BASE RES.	SPEEDUP (STEP TIME)
CIFAR-10	Conv2D	-	base	91.90%	0%
	S4ND	-	base	93.40%	0%
	Conv2D	80 – 20	low \rightarrow base	90.94%	51.7%
	S4ND	80 – 20	low \rightarrow base	92.32%	21.8%
CelebA	Conv2D	-	base	91.44%	0%
	S4ND	-	base	91.75%	0%
	Conv2D	16 – 4	low \rightarrow mid	80.89%	76.7%
	S4ND	16 – 4	low \rightarrow mid	88.57%	57.3%

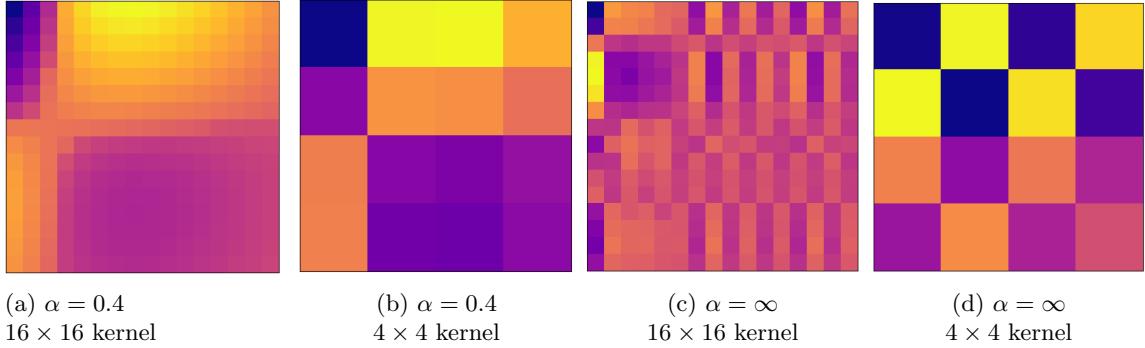


Figure 9.5: (**Effect of bandlimiting on learned kernels.**) Bandlimiting significantly increases the smoothness of the kernels when resizing resolutions.

Effect of Bandlimiting. Bandlimiting in S4ND is critical to generalization at different resolutions. We analyze the effect of the bandlimiting parameter α on CIFAR-10 performance when doing zero-shot resolution change. We additionally vary the choice of basis function $K_n(t)$ used in the SSM. Figure 9.3 shows that zero-shot performance on base degrades for larger values of α , i.e. for cutoffs that do not remove high frequencies that violate the Nyquist cutoff. As we would expect, this holds regardless of the choice of basis function. In Figure 9.5, we visualize learned kernels with and without bandlimiting, showing that bandlimiting improves smoothness. Appendix I.2.3 includes additional experiments that analyze α on Celeb-A, and when doing progressively resized training.

Chapter 10

Conclusion

Sequence models continue to drive major advances in deep learning, as the foundation of recent developments (e.g. the GPT family [149]) which have shown potential for transformative impact on machine learning, science, and human society at large. Yet there is still much room for new tools and building blocks that can be applied across applications and modalities with a broad range of capabilities. This thesis makes the argument that S4 models are a promising candidate, as a fundamental model with many properties, strong efficiency, and principled theory. To start, we explored the elegant representations of state space models as continuous, recurrent, and convolutional models, and their deep relationships with many previous families of sequence models (Chapter 2). We then discussed their drawbacks and designed new representations and algorithms to compute several classes of structured SSMs very efficiently (Chapter 3). We further formulated theoretical frameworks for learning principled memory representations with these models (Chapters 4 and 5), which combined neatly with our structured representations for the best of both worlds (Chapter 6). After developing the theory of S4, we validated its effectiveness as a general sequence modeling solution on established benchmarks across a diverse range of data modalities and model capabilities (Chapter 7). Finally, we showed how these capabilities such as fast autoregressive generation and zero-shot resolution adaptation extended to larger-scale and concrete applications on common continuous-signal data such as audio, images, and videos (Chapters 8 and 9).

As is common with new research areas, our progress presents as many questions as it does answers. We discuss some of these open questions and highlight broad future directions that we are most excited about, many of which have already seen follow-up work.

10.1 Core Model Understanding and Improvements

While progress in model architectures and core mechanisms has slowed down in recent years, particularly since the advent of the Transformer [217], we hope that our development of new model families and abilities opens up new directions. Much more remains both to understand and improve these models.

State Space Models

Several follow-up works have simplified, improved, and extended the core S4 formulation in several directions, some of which were concurrent to the developments in this thesis. These include DSS which first empirically evaluated diagonal S4 models [76], S5 which presents a MIMO SSM simplification [192], Liquid S4 which generalizes the linear SSM dynamics [79], and the exponential moving average (EMA) layer from Mega, which found that a simple real-valued S4D variant works empirically well on many settings, especially when combined with attention [134]. We find this last phenomenon particularly intriguing, which raises the problem of finding a theoretical explanation.

Recurrent Neural Networks

Our line of work originated with motivations from an RNN perspective, and we are still most excited about this representation. In many ways, it is strictly more flexible than the convolutional view, which is derived from restricted (LTI) forms of the recurrent representation. The Linear Recurrent Unit (LRU) provides nice simplifications and explanations of the mechanics of S4 from the perspective of designing a ground-up linear RNN [150]. We are particularly fond of the continuous SSM’s connection to gating mechanisms (Lemma 2.16), which remains one way in which time-invariant SSMs still do not generalize RNNs and we believe can lead to potential major improvements for this model family.

Convolutional Neural Networks

S4 introduced the view of long-kernel CNNs, developed concurrently by CKConv and FlexConv [174, 173], which has become one of the most popular interpretations in following work. Several models develop alternate parameterizations of implicit long convolution kernels, including SGConv [124], LongConv [64], Hyena [163], and MultiresConv [188]. A commonality of these models is the empirical observation of needing to enforce an exponentially-decaying prior, which is automatically enforced by SSM-parameterized long convolutions.

From one perspective this is counterintuitive, since exponential decay leads to weaker long-range dependencies, and a theoretical justification for this phenomenon—perhaps from an optimization perspective—remains open.

Combination with Attention

A major theme of this work is that S4 models inherit the best of both worlds from RNNs and CNNs; we do believe that they can become drop-in replacements more broadly. However, we believe that S4 and attention have quite different and complementary mechanisms, and can be combined effectively. This has been validated through a multitude of new neural network architectures (which could be considered SSNNs) combining attention and S4 variants, including GSS [136], Mega [134], H3 [37], SPADE [244], MSSM [58], DSSformer [183], and ViS4mer [93]. These models have started obtaining SOTA results on many data modalities including language, speech, and videos, raising questions on understanding and improving the empirical effectiveness of combining S4 and attention.

Steps can also be taken to unify the core mechanisms of RNNs and attention, which was pioneered by Linear Attention (LA) [104] and continued in recent mechanisms that can be interpreted as hybrids of SSMs and LA such as AFT [234], RWKV [158], and H3 [37]. This offers a rich avenue toward continuing to improve recurrent models.

Combining Modeling Mechanisms

Like convolutions and attention, S4 can be viewed as a simple sequence transformation that can be incorporated flexibly into models in many other ways. Some examples so far include combinations with graph neural networks [201], latent variable models [242], and diffusion models [3]. An enormous number of other modeling mechanisms exist such as GANs and normalizing flows, neural ODEs, energy-based mechanisms, etc. (and any combinations thereof), which could also be combined with state space models.

Beyond Fixed Context

Another primary motivation for the development of this work is the problem of building long-range models. An inherit limitation of most sequence models such as Transformers and CNNs is a fixed context window, while RNNs traditionally suffer from optimization issues that limit their range. Overcoming this bottleneck is still a fundamental unsolved problem. For example, an enormous array of efficient attention mechanisms have been developed to improve their quadratic scaling in the sequence length [203], but these only improve

constant factors that cannot extend the window indefinitely the way biological agents such as humans do. We believe that the methods developed here present useful tools and ideas toward addressing this intriguing problem.

10.2 Application Areas

We showed that state space models have many properties and capabilities that make them useful as components of models for a variety of applications. We are excited for many more potential applications, including but not limited to some of the possibilities below.

Language

Language is perhaps at the forefront of machine learning and artificial intelligence today. Interestingly, we found that straightforward applications of S4—while better than prior recurrence- and convolution-based models—do not perform as well as models based on attention. This is perhaps not surprising given language’s lack of interpretation as a continuous signal (which also applies to other important sequence data such as genomics), in contrast to the other modalities discussed below. Nevertheless, this creates opportunities to understand this phenomenon and improve attention-free models for discrete tokenized data. Some works have made substantial progress in this direction by designing better SSNN architectures, such as H3 [37] and Hyena [163] for autoregressive language modeling and BiGS [220] for masked language modeling. Related recurrent models such as RWKV [158] can also be interpreted as an SSM variant and have already shown the potential to scale as well as Transformers to the size of 14B parameters and beyond. As language is incredibly important for modern deep learning, making progress in this direction can have outsized impact, such as by significantly extending the context window of large language models.

Audio, Images, Videos

These modalities can generate massive amounts of data with long-term dependencies (e.g. for audio and videos), and are based on physical (continuous) signals that have a high affinity with S4’s inductive bias. Beyond our progress in Part III of this thesis, more results have validated the empirical effectiveness of this direction [93, 183, 58]. Particularly interesting is finding opportunities to take advantage of properties of continuous models such as robustness to spatial resolution and audio/video sampling rates, or working with extremely high resolution data such as medical images [1].

Large-scale Pretraining

Going hand-in-hand with the above modalities is the idea of large-scale pretrained models. For example, recent video benchmark datasets are significantly larger than the HMDB-51 dataset used in our experiments [105, 69, 35, 139, 190]. S4’s properties and computational scalability may lend itself well to being incorporated into the backbone of such foundation models [16], which is yet unexplored.

Time Series

Time series are a broad class of very heterogeneous data with many specialized approaches and models, that are particularly important for financial and industrial applications. Deep learning is commonly perceived to not perform well for time series [233], but it is interesting to explore whether the development of recent tools can facilitate deep learning’s breakthrough for this modality as AlexNet did for vision and the Transformer did for NLP. Progress has been made in the form of generation, imputation, and forecasting models such as Latent-S4 [242], SSSD [3], and SpaceTime [239], although we believe time series do still present fundamental and interesting challenges, such as its heterogeneity and non-stationarity.

Reinforcement Learning and Robotics

Reinforcement learning, particularly in partially-observed environments, can be considered a setting that inherently involves recurrent updates of a model’s (or agent’s) state. For this reason it has been an area where RNNs have been commonly used (e.g. [102]) and are perhaps more suited than alternative sequence models. This presents another interesting opportunity to leverage stateful models that have both fast training and efficient rollouts.

Similarly, various applications in robotics share these characteristics; in fact, state space models were first invented in the adjacent field of controls [101]. This is also another area where continuous models may be useful, e.g. to handle multiple input data streams at different sampling rates.

10.3 Hardware Efficiency

This work develops new algorithms for S4 models that lets them be reasonably efficient on modern accelerators like GPUs. However, the efficiency of these models are still somewhat

limited compared to mechanisms such as local convolutions and attention, which are based on dense matrix multiplications that these accelerators have been highly specialized for. This has led to an accelerating rate of progress for models such as CNNs and Transformers: they were popular in part because of efficiency and scalability, leading to more specialized hardware for matrix multiplications, feeding back into dedicated research focusing on these models. This phenomenon has sometimes been called the *hardware lottery* [90].

As the central theme of this thesis, we believe that there is tremendous potential for completely different mechanisms to be effective. However, these would benefit from their own software-hardware feedback loop. S4 in particular requires either of two non-standard mechanisms: either (i) fast sequential recurrences, or (ii) fast FFT (Fast Fourier Transform) operations for the convolutional view. Both of these are memory-bound operations (i.e. the runtime is dominated by the time to read/write to GPU memory) that are quite suboptimal on modern accelerators. For example, benchmarking the S4ND model found that the majority of time (over 60%) was spent in the FFTs. A dedicated implementation that treats S4 as a primitive could fuse the kernels (i.e. load the input once, perform all arithmetic operations, then write the final result back to memory) to potentially increase its speed by several times. Such hardware-aware optimizations have shown tremendous speed-ups for other operations such as FlashAttention [36].

Even so, S4 models fundamentally do not involve large matrix multiplications, which inherently limits them by the hardware lottery; even if their FLOP to performance tradeoff is better than other sequence modeling mechanisms, these FLOPs are in a sense more expensive. In order to realize the full potential of these models, there are opportunities for lower-level system and even hardware optimizations that target these primitives (for example, perhaps optimizing for recurrence via pipeline parallelism over the sequence length).

It is an incredibly exciting period for deep learning research, with constant rapid advances that have considerably started to impact the real world. We hope that our development of new models inspires more progress in sequence modeling mechanisms, opens up new application and research areas, and perhaps most importantly, has been an intrinsically interesting and elegant subject.

Part IV

Appendices

Appendix A

Technical Preliminaries

A.1 Discretization

In our setting, time series data are modeled by a continuous function and our models are (continuous-time) differential equations; they must be discretized to work on sequences. We provide a self-contained background and derivations of several ODE discretization methods, which are used both to convert continuous-time models to discrete-time used throughout this thesis, and are also used in our results on understanding RNNs (Section 2.4.2).

We first introduce two standard approximation schemes for differential equations that we will use to convert continuous-time models to discrete-time, and will be used in our results on understanding RNNs.

A.1.1 Overview

We consider the standard setting of a first-order initial value problem (IVP) ordinary differential equation (ODE) for a continuous function $f(t, x)$

$$\begin{aligned}\dot{x}(t) &= f(t, x(t)) \\ x(t_0) &= x_0\end{aligned}. \tag{A.1}$$

This differential form has an equivalent integral form

$$x(t) = x_0 + \int_{t_0}^t f(s, x(s)) ds. \tag{A.2}$$

At a high level, the basic numerical integration techniques considered here use the integral form (A.2) and use simple approximations of the RHS integral to iterately generate an approximate solution for x . For example, *Picard iteration* is often used to prove the existence of solutions to ODEs by iterating the equation $x_{i+1}(t) := x_i(t_0) + \int_{t_0}^t f(s, x_i(s)) ds$. In other words, it finds a sequence of functions $x_0(t), x_1(t), \dots$ that approximate the solution $x(t)$ of the integral equation. On the other hand, for a desired sequence of discrete times t_i , approximations to $x(t_0), x(t_1), \dots$ can be found by iterating the equation $x(t_{i+1}) = x(t_i) + \int_{t_i}^{t_{i+1}} f(s, x(s)) ds$. Different ways of approximating the RHS integral lead to different *discretization* schemes.

Appendices A.1.2 and A.1.3 overview the Picard theorem and first-order numerical integration methods, which apply to any IVP (A.1). Appendix A.1.4 then shows how to specialize it to linear time-invariant (LTI) systems as in state space models (2.1). Appendix A.1.5 derives a new discretization rule for “linear scale-invariant” (LSI) systems such as HIPPO-LegS (Section 4.3).

Empirical results comparing the effects of discretizations can be found throughout this thesis. For example, Section 4.4.2 provides a simple ablation of comparing the errors between three discretization rules applied to the HIPPO operators from Chapter 4.

A.1.2 Picard Iteration

The **Picard-Lindelöf Theorem** gives sufficient conditions for the existence and uniqueness of solutions to an IVP. As part of the proof, it provides an iteration scheme to compute this solution.

Theorem A.1 (Picard-Lindelöf). *In the IVP (A.1), if there is an interval around t_0 such that f is Lipschitz in its second argument, then there is an open interval $I \ni t_0$ such that there exists a unique solution $x(t)$ to the IVP in I . Furthermore, the sequence of **Picard iterates** $x^{(0)}, x^{(1)}, \dots$ converges to x , defined iteratively by*

$$\begin{aligned} x^{(0)}(t) &= x_0 \\ x^{(\ell)}(t) &= x_0 + \int_{t_0}^t f(s, x^{(\ell-1)}(s)) ds. \end{aligned}$$

The Picard iteration can be viewed as approximating (A.2) by holding the previous estimate of the solution $x^{(\ell-1)}$ fixed inside the RHS integral.

A.1.3 Numerical Integration Methods

Many methods for numerical integration of ODEs exist, which calculate discrete-time approximations of the solution. We discuss a few of the simplest methods, which are first-order methods with local error $O(h^2)$ [19].

These methods start by discretizing (A.2) into the form

$$x(t_k) - x(t_{k-1}) = \int_{t_{k-1}}^{t_k} f(s, x(s)) ds. \quad (\text{A.3})$$

Here we assume a sequence of discrete times t_0, t_1, t_2, \dots is fixed. For convenience, let x_k denote $x(t_k)$ and let $\Delta_k := t_k - t_{k-1}$. The goal is now to approximate the integral in the RHS of (A.3).

Euler method. The Euler method approximates (A.3) by holding the left endpoint constant throughout the integral (i.e., the “rectangle rule” with left endpoint), $f(s, x(s)) \approx f(t_{k-1}, x(t_{k-1}))$. The discrete-time update becomes

$$\begin{aligned} x_k - x_{k-1} &= (t_k - t_{k-1})f(t_{k-1}, x(t_{k-1})) \\ &= \Delta_k f(t_{k-1}, x_{k-1}). \end{aligned} \quad (\text{A.4})$$

Backward Euler method. The backward Euler method approximates (A.3) by holding the right endpoint constant throughout the integral (i.e., the “rectangle rule” with right endpoint), $f(s, x(s)) \approx f(t_k, x(t_k))$. The discrete-time update becomes

$$\begin{aligned} x_k - x_{k-1} &= (t_k - t_{k-1})f(t_k, x(t_k)) \\ &= \Delta_k f(t_k, x_k). \end{aligned} \quad (\text{A.5})$$

A.1.4 Discretization of LTI Systems (T-SSMs)

In the case of a linear system, the IVP is specialized to the case

$$x'(t) = f(t, x(t)) = \mathbf{A}x(t) + \mathbf{B}u(t).$$

Note that here u is treated as a fixed external input, which is constant from the point of view of this ODE in x .

The technique of approximating the integral equation (A.3) can be specialized to this case.

Since this LTI system is simpler than a general ODE, more specific approximation formulas can be derived.

Generalized Bilinear Transform Generalizing the Euler and Backwards Euler methods above, a convex combination of the left and right endpoints can be taken to approximate the integral, weighing them by $1 - \alpha$ and α respectively.

Let u_k denote the average value in each discrete time interval,

$$u_k = \frac{1}{\Delta_k} \int_{t_{k-1}}^{t_k} u(s) ds.$$

Note that the case $\alpha = 0, 1$ are specializations of the forward and backward Euler method, and the case $\alpha = \frac{1}{2}$ is the classic “trapezoid rule” for numerical integration.

$$\begin{aligned} x(t_k) - x(t_{k-1}) &= \int_{t_{k-1}}^{t_k} \mathbf{A}x(s) ds + \int_{t_{k-1}}^{t_k} \mathbf{B}u(s) ds \\ &= \int_{t_{k-1}}^{t_k} \mathbf{A}x(s) ds + \Delta_k \mathbf{B}u_k \\ &\approx \Delta_k [(1 - \alpha)\mathbf{A}x_{k-1} + \alpha\mathbf{A}x_k] + \Delta_k \mathbf{B}u_k. \end{aligned}$$

Rearranging yields

$$\begin{aligned} (\mathbf{I} - \alpha\Delta_k \cdot \mathbf{A})x_k &= (\mathbf{I} + (1 - \alpha)\Delta_k \cdot \mathbf{A})x_{k-1} + \Delta_k \cdot \mathbf{B}u_k \\ x_k &= (\mathbf{I} - \alpha\Delta_k \cdot \mathbf{A})^{-1}(\mathbf{I} + (1 - \alpha)\Delta_k \cdot \mathbf{A})x_{k-1} + (\mathbf{I} - \alpha\Delta_k \cdot \mathbf{A})^{-1}\Delta_k \cdot \mathbf{B}u_k \end{aligned}$$

This derives the **generalized bilinear transform (GBT)** [236]. The **bilinear method** is the case $\alpha = \frac{1}{2}$ of special significance, and was numerically found to be better than the forward and backward Euler methods $\alpha = 0, 1$ both in synthetic function approximation settings and in end-to-end experiments [70, Figure 4].

For completeness and ease of reference, we write out these discretization formulas both in sequence and function notation.

Forward Euler

$$\begin{aligned}x_k &= (\mathbf{I} + \Delta_k \mathbf{A})x_{k-1} + \Delta_k \mathbf{B} \cdot u_k \\x(t + \Delta) &= (\mathbf{I} + \Delta \mathbf{A})x(t) + \Delta \mathbf{B} \cdot u(t)\end{aligned}$$

Backward Euler

$$\begin{aligned}x_k &= (\mathbf{I} - \Delta_k \mathbf{A})^{-1}x_{k-1} + (\mathbf{I} - \Delta_k \mathbf{A})^{-1}\Delta_k \mathbf{B} \cdot u_k \\x(t + \Delta) &= (\mathbf{I} - \Delta \mathbf{A})^{-1}x(t) + (\mathbf{I} - \Delta \mathbf{A})^{-1}\Delta \mathbf{B} \cdot u(t)\end{aligned}$$

Bilinear (a.k.a. Trapezoid rule, a.k.a. Tustin's method)

$$\begin{aligned}x_k &= \left(\mathbf{I} - \frac{\Delta_k}{2} \mathbf{A}\right)^{-1} \left(\mathbf{I} + \frac{\Delta_k}{2} \mathbf{A}\right) \cdot x_{k-1} + \left(\mathbf{I} - \frac{\Delta_k}{2} \mathbf{A}\right)^{-1} \Delta_k \mathbf{B} \cdot u_k \\x(t + \Delta) &= \left(\mathbf{I} - \frac{\Delta}{2} \mathbf{A}\right)^{-1} \left(\mathbf{I} + \frac{\Delta}{2} \mathbf{A}\right) \cdot x(t) + \left(\mathbf{I} - \frac{\Delta}{2} \mathbf{A}\right)^{-1} \Delta \mathbf{B} \cdot u(t)\end{aligned}$$

Zero-Order Hold (ZOH) Finally, the zero-order hold discretization is specialized to LTI systems and uses a simple approximation where the system input $u(t)$ is assumed to be constant between time t and $t + \Delta$. This yields the update

$$\begin{aligned}x_k &= e^{\Delta_k \mathbf{A}}x_{k-1} + (\Delta_k \mathbf{A})^{-1}(e^{\Delta_k \mathbf{A}} - \mathbf{I}) \cdot \Delta_k \mathbf{B} \cdot u_k \\x(t + \Delta) &= e^{\Delta \mathbf{A}}x(t) + (\Delta \mathbf{A})^{-1}(e^{\Delta \mathbf{A}} - \mathbf{I}) \cdot \Delta \mathbf{B} \cdot u(t)\end{aligned}$$

Note that this form is written in a way that shows that the discrete updates depend only on ΔA and ΔB (Lemma 2.13).

A.1.5 Discretization of Linear Scale Invariant (LSI) Systems

In the case of HIPPO-LegS (Section 4.3), we have a linear ODE of the form $x'(t) = \frac{1}{t} \mathbf{A}x(t) + \frac{1}{t} \mathbf{B}u(t)$. We call this an LSI system (Definition 4.4), which have special properties, such as being invariant to the rate of evolution of the system. As a consequence, discretizing this system at any rate Δ gives the same discrete dynamics.

Adapting the GBT discretization (which generalizes forward/backward Euler and bilinear)

to this linear ODE, we obtain:

$$x(t + \Delta) - \Delta\alpha \frac{1}{t + \Delta} \mathbf{A}x(t + \Delta) = \left(\mathbf{I} + \Delta(1 - \alpha) \frac{1}{t} \mathbf{A} \right) x(t) + \Delta \frac{1}{t} \mathbf{B}u(t)$$

and solving for $x(t + \Delta)$ gives

$$x(t + \Delta) = \left(\mathbf{I} - \frac{\Delta}{t + \Delta} \alpha \mathbf{A} \right)^{-1} \left(\mathbf{I} + \frac{\Delta}{t} (1 - \alpha) \mathbf{A} \right) x(t) + \frac{\Delta}{t} \left(\mathbf{I} - \frac{\Delta}{t + \Delta} \alpha \mathbf{A} \right)^{-1} \mathbf{B}u(t).$$

We highlight that this system is invariant to the discretization step size Δ . Indeed, if $x_k := x(k\Delta)$ and $u_k := u(k\Delta)$ then we have the recurrence

$$x_{k+1} = \left(\mathbf{I} - \frac{1}{k+1} \alpha \mathbf{A} \right)^{-1} \left(\mathbf{I} + \frac{1}{k} (1 - \alpha) \mathbf{A} \right) x_k + \frac{1}{k} \left(\mathbf{I} - \frac{1}{k+1} \alpha \mathbf{A} \right)^{-1} \mathbf{B}u_k,$$

which does not depend at all on Δ .

A.2 Orthogonal Polynomials

Orthogonal polynomials are a standard tool for working with function spaces [26, 197]. Every measure μ induces a unique (up to a scalar) sequence of *orthogonal polynomials* (OPs) $P_0(x), P_1(x), \dots$ satisfying $\deg(P_i) = i$ and $\langle P_i, P_j \rangle_\mu := \int P_i(x)P_j(x) d\mu(x) = 0$ for all $i \neq j$. This is the sequence found by orthogonalizing the monomial basis $\{x^i\}$ with Gram-Schmidt with respect to $\langle \cdot, \cdot \rangle_\mu$. The fact that OPs form an orthogonal basis is useful because the optimal polynomial g of degree $\deg(g) < N$ that approximates a function f is then given by

$$\sum_{i=0}^{N-1} c_i P_i(x) / \|P_i\|_\mu^2 \quad \text{where } c_i = \langle f, P_i \rangle_\mu = \int f(x)P_i(x) d\mu(x).$$

Classical OPs families comprise Jacobi (which include Legendre and Chebyshev polynomials as special cases), Laguerre, and Hermite polynomials. The Fourier basis can also be interpreted as OPs on the unit circle in the complex plane.

A.2.1 Properties of Legendre Polynomials

Legendre Polynomials

Under the usual definition of the canonical Legendre polynomial P_n , they are orthogonal with respect to the measure $\omega^{\text{leg}} = \mathbf{1}_{[-1,1]}$:

$$\frac{2n+1}{2} \int_{-1}^1 P_n(x) P_m(x) dx = \delta_{nm} \quad (\text{A.6})$$

Also, they satisfy

$$\begin{aligned} P_n(1) &= 1 \\ P_n(-1) &= (-1)^n. \end{aligned}$$

Shifted and Scaled Legendre Polynomials

We will also consider scaling the Legendre polynomials to be orthogonal on the interval $[0, t]$. A change of variables on (A.6) yields

$$\begin{aligned} &(2n+1) \int_0^t P_n\left(\frac{2x}{t} - 1\right) P_m\left(\frac{2x}{t} - 1\right) \frac{1}{t} dx \\ &= (2n+1) \int P_n\left(\frac{2x}{t} - 1\right) P_m\left(\frac{2x}{t} - 1\right) \omega^{\text{leg}}\left(\frac{2x}{t} - 1\right) \frac{1}{t} dx \\ &= \frac{2n+1}{2} \int P_n(x) P_m(x) \omega^{\text{leg}}(x) dx \\ &= \delta_{nm}. \end{aligned}$$

Therefore, with respect to the measure $\omega_t = \mathbf{1}_{[0,t]}/t$ (which is a probability measure for all t), the normalized orthogonal polynomials are

$$(2n+1)^{1/2} P_n\left(\frac{2x}{t} - 1\right).$$

Similarly, the basis

$$(2n+1)^{1/2} P_n\left(2\frac{x-t}{\theta} + 1\right)$$

is orthonormal for the uniform measure $\frac{1}{\theta} \mathbb{I}_{[t-\theta, t]}$.

In general, the orthonormal basis for any uniform measure consists of $(2n+1)^{\frac{1}{2}}$ times the corresponding linearly shifted version of P_n .

Derivatives of Legendre Polynomials

We note the following recurrence relations on Legendre polynomials ([4, Chapter 12]):

$$\begin{aligned} (2n+1)P_n &= P'_{n+1} - P'_{n-1} \\ P'_{n+1} &= (n+1)P_n + xP'_n \end{aligned}$$

The first equation yields

$$P'_{n+1} = (2n+1)P_n + (2n-3)P_{n-2} + \dots, \quad (\text{A.7})$$

where the sum stops at P_0 or P_1 .

These equations directly imply

$$P'_n = (2n-1)P_{n-1} + (2n-5)P_{n-3} + \dots \quad (\text{A.8})$$

and

$$\begin{aligned} (x+1)P'_n(x) &= P'_{n+1} + P'_n - (n+1)P_n \\ &= nP_n + (2n-1)P_{n-1} + (2n-3)P_{n-2} + \dots \end{aligned} \quad (\text{A.9})$$

These will be used in the derivations of the HIPPO-LegT and HIPPO-LegS updates, respectively.

A.3 Other Mathematical Identities

A.3.1 Leibniz Integral Rule

As part of our standard strategy for deriving HIPPO update rules (Appendix D.1), we will differentiate through integrals with changing limits. For example, we may wish to differentiate with respect to t the expression $\int f(t, x)\mu(t, x) dx = \int_0^t f(t, x)\frac{1}{t} dx$ when analyzing the scaled Legendre (LegS) measure.

Differentiating through such integrals can be formalized by the Leibniz integral rule, the

basic version of which states that

$$\frac{\partial}{\partial t} \int_{\alpha(t)}^{\beta(t)} f(x, t) dx = \int_{\alpha(t)}^{\beta(t)} \frac{\partial}{\partial t} f(x, t) dx - \alpha'(t)f(\alpha(t), t) + \beta'(t)f(\beta(t), t).$$

We elide over the formalisms in our derivations (Appendix D.2) and instead use the following trick. We replace integrand limits with an indicator function; and using the Dirac delta function δ when differentiating (i.e., using the formalism of distributional derivatives). For example, the above formula can be derived succinctly with this trick:

$$\begin{aligned} \frac{\partial}{\partial t} \int_{\alpha(t)}^{\beta(t)} f(x, t) dx &= \frac{\partial}{\partial t} \int f(x, t) \mathbb{I}_{[\alpha(t), \beta(t)]}(x) dx \\ &= \int \frac{\partial}{\partial t} f(x, t) \mathbb{I}_{[\alpha(t), \beta(t)]}(x) dx + \int f(x, t) \frac{\partial}{\partial t} \mathbb{I}_{[\alpha(t), \beta(t)]}(x) dx \\ &= \int \frac{\partial}{\partial t} f(x, t) \mathbb{I}_{[\alpha(t), \beta(t)]}(x) dx + \int f(x, t) (\beta'(t)\delta_{\beta(t)}(x) - \alpha'(t)\delta_{\alpha(t)}(x)) dx \\ &= \int_{\alpha(t)}^{\beta(t)} \frac{\partial}{\partial t} f(x, t) dx - \alpha'(t)f(\alpha(t), t) + \beta'(t)f(\beta(t), t). \end{aligned}$$

A.3.2 Woodbury Matrix Identity

The Woodbury identity states that the inverse of a low-rank perturbation to a matrix, is a low-rank perturbation of the matrix's inverse. We leverage it in multiple efficient algorithms for SSMs, including DPLR SSM kernels (Section 3.3) and an earlier version for quasiseparable SSM kernels (Section 6.1).

Proposition A.2 (Binomial Inverse Theorem or Woodbury matrix identity [225, 68]). *Over a commutative ring \mathcal{R} , let $\mathbf{A} \in \mathcal{R}^{N \times N}$ and $\mathbf{U}, \mathbf{V} \in \mathcal{R}^{N \times p}$. Suppose \mathbf{A} and $\mathbf{A} + \mathbf{U}\mathbf{V}^*$ are invertible. Then $\mathbf{I}_p + \mathbf{V}^* \mathbf{A}^{-1} \mathbf{U} \in \mathcal{R}^{p \times p}$ is invertible and*

$$(\mathbf{A} + \mathbf{U}\mathbf{V}^*)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1} \mathbf{U} (\mathbf{I}_p + \mathbf{V}^* \mathbf{A}^{-1} \mathbf{U})^{-1} \mathbf{V}^* \mathbf{A}^{-1}$$

Appendix B

Appendix for Chapter 2

B.1 Proofs: RNNs are SSMs

This section refines the statements in Section 2.4; we give more detailed statements and proofs of Lemma 2.16 and Lemma 2.17 in Lemma B.1 and Lemma B.3, respectively.

Appendix A.1 has a more detailed (and self-contained) summary of basic methods in ODE approximation which will be used in the results and proofs.

Overall, these results point toward the connection that SSMs and popular families of RNN methods all approximate the same continuous-time dynamics

$$\dot{x}(t) = -x + f(t, x(t)) \quad (\text{B.1})$$

by viewing them with a combination of two techniques.

We note that these results are about two of the most commonly used architecture modifications for RNNs. First, the gating mechanism is ubiquitous in RNNs, and usually thought of as a heuristic for smoothing optimization [86]. Second, many of the effective large-scale RNNs use linear (gated) recurrences and deeper models, which is usually thought of as a heuristic for computational efficiency [17]. Our results suggest that neither of these are heuristics after all, and arise from standard ways to approximate ODEs.

To be more specific, we show that:

- Nonlinear RNNs discretize the dynamics (B.1) by applying backwards Euler discretization to the linear term, which arises in the gating mechanism of RNNs (Appendix B.1.2, Lemma B.1).
- A special case of deep SSMs approximates the dynamics (B.1) (in continuous-time) by applying Picard iteration to the nonlinear term (Appendix B.1.3, Lemma B.3).
- Deep linear RNNs approximate the dynamics (B.1) with both Picard iteration in the depth direction to linearize the nonlinear term, and discretization (gates) in the time direction to discretize the equation (Appendix B.1.4, Corollary B.4).

In the remainder of this section, we assume that there is an underlying function $x(t)$ that satisfies (B.1) on some interval for any initial condition, and that f is continuous and Lipschitz in its second argument. Our goal is to show that several families of models approximate this in various ways.

B.1.1 Intuition and Proof Sketches

We sketch the idea of how SSMs capture popular RNNs. More precisely, we will show how approximating the dynamics (B.1) in various ways lead to types of RNNs and SSMs.

The first step is to look at the simpler dynamics

$$x'(t) = -x(t) + u(t)$$

where there is some input $u(t)$ that is independent of x . (In other words, in (B.1), the function $f(t, x)$ does not depend on the second argument.)

By directly applying the a discretization (e.g. backwards Euler), this leads to a gated recurrence (Lemma 2.16).

The second step is that by applying the backwards Euler discretization more directly to (B.1), this leads to a gated RNN where the input can depend on the state (Lemma B.1).

Alternatively, we can apply Picard iteration on (B.1), which says that the iteration

$$x^{(\ell)}(t) = x_0 + \int_{t_0}^t -x^{(\ell-1)}(s) ds + \int_{t_0}^t f(s, x^{(\ell-1)}(s)) ds$$

converges to the solution $x(t)$.

However, the first integral term is simple and can be tightened. We can instead try to apply Picard iteration on only the second term, leaving the first integral in terms of $x^{(\ell)}$. Intuitively this should still converge to the right solution, since this is a weaker iteration; we're only using the Picard approximation on the second term. So we can replace $x^{(\ell-1)}$ with $x^{(\ell)}$ in the first term,

$$x^{(\ell)}(t) = x_0 + \int_{t_0}^t -x^{(\ell)}(s) ds + \int_{t_0}^t f(s, x^{(\ell-1)}(s)) ds.$$

Differentiating, this equation is the ODE

$$\dot{x}^{(\ell)}(t) = -x^{(\ell)}(t) + f(t, x^{(\ell-1)}(t))$$

This implies that alternating point-wise functions with a simple linear ODE $\dot{x}^{(\ell)}(t) = -x^{(\ell)}(t) + u^{(\ell)}(t)$ also captures the dynamics (B.1). But this is essentially what a deep SSM is.

To move to discrete-time, this continuous-time layer can be discretized with gates as in Lemma 2.16, leading to deep linear RNNs such as the QRNN, or with SSM discretizations, leading to the discrete-time SSM. We note again that in the discrete-time SSM, $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ play the role of the RNN gates $\sigma(z)$ and $1 - \sigma(z)$.

B.1.2 Capturing Gates through Discretization

Lemma B.1. *Consider an RNN of the form*

$$x_k = (1 - \sigma(z_k))x_{k-1} + \sigma(z_k)\bar{f}(k, x_{k-1}), \quad (\text{B.2})$$

where $\bar{f}(k, x)$ is an arbitrary function that is Lipschitz in its second argument (e.g., it may depend on an external input u_k).

Then equation (B.2) is a discretization of the dynamics (B.1) with step sizes $\Delta_k = \exp(z_k)$, i.e. $x_k \approx x(t_k)$ where $t_k = \sum_{i=1}^k \Delta_i$.

Proof. Apply the backwards Euler discretization (A.5) to equation (B.1) to get

$$\begin{aligned} x_k - x_{k-1} &= \Delta_k [-x_k + f(t_k, x_k)] \\ (1 + \Delta_k)x_k &= x_{k-1} + \Delta_k f(t_k, x_k) \\ x_k &= \frac{1}{1 + \Delta_k} x_{k-1} + \frac{\Delta_k}{1 + \Delta_k} f(t_k, x_k). \end{aligned}$$

Note that $\frac{\Delta_k}{1 + \Delta_k} = \frac{e^{z_k}}{1 + e^{z_k}} = \frac{1}{1 + e^{-z_k}}$ and $\frac{1}{1 + \Delta_k} = 1 - \frac{\Delta_k}{1 + \Delta_k}$, thus

$$x_k = (1 - \sigma(z_k))x_{k-1} + \sigma(z_k)\bar{f}(k, x_{k-1}).$$

Here we are denoting $\bar{f}(k, x) = f(t_k, x)$ to be a discrete-time version of f evaluable at the given timesteps t_k . \square

Note that a potential external input function $u(t)$ or sequence u_k is captured through the abstraction $f(t, x)$. For example, a standard vanilla RNN could define $\bar{f}(k, x) = f(t_k, x) = \tanh(\mathbf{W}x + \mathbf{U}u_k)$.

B.1.3 Capturing Nonlinear Dynamics through Picard Iteration

The main result of this section is Lemma B.3 showing that SSMs can approximate the same dynamics as the RNNs in the previous section. This follows from a technical lemma.

Lemma B.2. *Let $f(t, x)$ be any function that satisfies the conditions of the Picard-Lindelöf Theorem (Theorem A.1).*

Define a sequence of functions $x^{(\ell)}$ by alternating the (point-wise) function f with solving an ODE

$$\begin{aligned} x^{(0)}(t) &= x_0 \\ u^{(\ell)}(t) &= f(t, x^{(\ell-1)}(t)) \\ \dot{x}^{(\ell)}(t) &= \mathbf{A}x^{(\ell)}(t) + u^{(\ell)}(t). \end{aligned}$$

Then $x^{(\ell)}$ converges to a solution $x^{(\ell)}(t) \rightarrow x(t)$ of the IVP

$$\begin{aligned} \dot{x}(t) &= \mathbf{A}x(t) + f(t, x(t)) \\ x(t_0) &= x_0. \end{aligned}$$

Lemma B.3. *A (continuous-time) deep SSM with order $N = 1$ and $\mathbf{A} = -1, \mathbf{B} = 1, \mathbf{C} = 1, \mathbf{D} = 0$ approximates the nonlinear dynamics (B.1).*

Proof. Applying the definition of an SSM (equations (2.1a)+(2.1b)) with these parameters results in a layer mapping $u(t) \mapsto y(t)$ where y is defined implicitly through the ODE

$$\dot{y}(t) = -y(t) + u(t).$$

This can be seen since the choice of \mathbf{C}, \mathbf{D} implies $y(t) = x(t)$ and the choice of \mathbf{A}, \mathbf{B} gives the above equation.

Consider the deep SSM defined by alternating this SSM with position-wise (in time) nonlinear functions

$$\begin{aligned} u^{(\ell)}(t) &= f(t, y^{(\ell-1)}(t)) \\ \dot{y}^{(\ell)}(t) &= -y^{(\ell)}(t) + u^{(\ell)}(t). \end{aligned}$$

But this is exactly a special case of Lemma B.2, so that we know $y^{(\ell)}(t) \rightarrow y(t)$ such that $y(t)$ satisfies

$$\dot{y}(t) = -y(t) + f(t, y(t))$$

as desired. \square

Proof of Lemma B.2. Let

$$z(t) = e^{-\mathbf{A}t}x(t)$$

(and $z_0 = z(t_0) = x(t_0) = x_0$). Note that

$$\begin{aligned} \dot{z}(t) &= e^{-\mathbf{A}t}[\dot{x}(t) - \mathbf{A}x(t)] \\ &= e^{-\mathbf{A}t}f(t, x(t)) \\ &= e^{-\mathbf{A}t}f(t, e^{\mathbf{A}t}z(t)). \end{aligned}$$

Since f satisfies the conditions of the Picard Theorem (i.e., is continuous in the first argument and Lipschitz in the second), so does the function g where $g(t, x) := e^{-\mathbf{A}t}f(t, e^{\mathbf{A}t}x)$ for some interval around the initial time.

By Theorem A.1, the iterates $z^{(\ell)}$ defined by

$$z^{(\ell)}(t) = z_0 + \int_{t_0}^t e^{-\mathbf{A}s} f(s, e^{\mathbf{A}s} z^{(\ell-1)}(s)) ds \quad (\text{B.3})$$

converges to z .

Define $x^{(\ell)}(t) = e^{\mathbf{A}t} z^{(\ell)}(t)$. Differentiate (B.3) to get

$$\begin{aligned} \dot{z}^{(\ell)}(t) &= e^{-\mathbf{A}t} f(t, e^{\mathbf{A}t} z^{(\ell-1)}(t)) \\ &= e^{-\mathbf{A}t} f(t, x^{(\ell-1)}(t)) \\ &= e^{-\mathbf{A}t} u^{(\ell)}(t). \end{aligned}$$

But

$$\dot{z}^{(\ell)}(t) = e^{-\mathbf{A}t} \left[\dot{x}^{(\ell)}(t) - \mathbf{A}x^{(\ell)}(t) \right],$$

so

$$\dot{x}^{(\ell)}(t) = \mathbf{A}x^{(\ell)}(t) + u^{(\ell)}(t).$$

Since $z^{(\ell)} \rightarrow z$ and $x^{(\ell)}(t) = e^{\mathbf{A}t} z^{(\ell)}(t)$ and $x(t) = e^{\mathbf{A}t} z(t)$, we have $x^{(\ell)} \rightarrow x$. \square

B.1.4 Capturing Deep, Linear, Gated RNNs

We finally note that several types of RNNs exist which were originally motivated by approximating linearizing gated RNNs for speed. Although these were treated as a heuristic for efficiency reasons, they are explained by combining our two main technical results.

Lemma B.1 shows that a single-layer, discrete-time, nonlinear RNN approximates the dynamics (B.1) through discretization, which arises in the gating mechanism.

Lemma B.3 shows that a deep, continuous-time, linear RNN approximates (B.1) through Picard iteration, where the nonlinearity is moved to the depth direction.

Combining these two results leads to Corollary B.4, which says that a deep, discrete-time, linear RNN can also approximate the same dynamics (B.1).

Corollary B.4. Consider a deep, linear RNN of the form

$$\begin{aligned} x_k^{(\ell)} &= (1 - \sigma(z_k))x_{k-1}^{(\ell)} + \sigma(z_k)u_k^{(\ell)} \\ u_k^{(\ell)} &= \bar{f}(k, x_k^{(\ell-1)}). \end{aligned}$$

This is a discretization of the dynamics (B.1) with step sizes $\Delta_k = \exp(z_k)$, i.e. $x_k \approx x(t_k)$ where $t_k = \sum_{i=1}^k \Delta_i$.

Proof. By Lemma B.1, the first equation is a discretization of the continuous-time equation

$$\dot{x}^{(\ell)}(t) = -x^{(\ell)}(t) + u^{(\ell)}(t)$$

where

$$u^{(\ell)}(t) = f(t, x^{(\ell-1)}(t))$$

uses the continuous-time version f of \bar{f} . But by Lemma B.2, this is an approximation of the dynamics (B.1) using Picard iteration. \square

Notable examples of this type of model include the Quasi-RNN or QRNN [17] and the Simple Recurrent Unit (SRU) [120], which are among the most effective models in practice. We remark that these are the closest models to the SSM and suggest that their efficacy is a consequence of the results of this section, which shows that they are not heuristics.

We note that there are many more RNN variants that use a combination of these gating and linearization techniques that were not mentioned in this section, and can be explained similarly.

Appendix C

Appendix for Chapter 3

C.1 S4-DPLR Algorithm Details

This section proves the results of Section 3.3.2, providing complete details of our efficient algorithms for S4.

Appendices C.1.1, C.1.2 and F.2.2 prove Theorems 3.5, 3.6 and 6.7 respectively.

C.1.1 Computing the S4 Recurrent View

We prove Theorem 3.5 showing the efficiency of the S4 parameterization for computing one step of the recurrent representation (Section 2.3.2).

Recall that without loss of generality, we can assume that the state matrix $\mathbf{A} = \boldsymbol{\Lambda} - \mathbf{P}\mathbf{Q}^*$ is diagonal plus low-rank (DPLR), potentially over \mathbb{C} . Our goal in this section is to explicitly write out a closed form for the (bilinear) discretized matrix $\overline{\mathbf{A}}$.

Recall from equation (2.5) that

$$\begin{aligned}\overline{\mathbf{A}} &= (\mathbf{I} - \Delta/2 \cdot \mathbf{A})^{-1}(\mathbf{I} + \Delta/2 \cdot \mathbf{A}) \\ \overline{\mathbf{B}} &= (\mathbf{I} - \Delta/2 \cdot \mathbf{A})^{-1}\Delta\mathbf{B}.\end{aligned}$$

We first simplify both terms in the definition of $\overline{\mathbf{A}}$ independently.

Forward Discretization

The first term is essentially the Euler discretization motivated in Section 2.3.1.

$$\begin{aligned}\mathbf{I} + \frac{\Delta}{2}\mathbf{A} &= \mathbf{I} + \frac{\Delta}{2}(\boldsymbol{\Lambda} - \mathbf{P}\mathbf{Q}^*) \\ &= \frac{\Delta}{2} \left[\frac{2}{\Delta}\mathbf{I} + (\boldsymbol{\Lambda} - \mathbf{P}\mathbf{Q}^*) \right] \\ &= \frac{\Delta}{2}\mathbf{A}_0\end{aligned}$$

where \mathbf{A}_0 is defined as the term in the final brackets.

Backward Discretization

The second term is known as the Backward Euler's method. Although this inverse term is normally difficult to deal with, in the DPLR case we can simplify it using Woodbury's Identity (Proposition A.2).

$$\begin{aligned}\left(\mathbf{I} - \frac{\Delta}{2}\mathbf{A}\right)^{-1} &= \left(\mathbf{I} - \frac{\Delta}{2}(\boldsymbol{\Lambda} - \mathbf{P}\mathbf{Q}^*)\right)^{-1} \\ &= \frac{2}{\Delta} \left[\frac{2}{\Delta} - \boldsymbol{\Lambda} + \mathbf{P}\mathbf{Q}^* \right]^{-1} \\ &= \frac{2}{\Delta} \left[\mathbf{D} - \mathbf{D}\mathbf{P}(\mathbf{I} + \mathbf{Q}^*\mathbf{D}\mathbf{P})^{-1}\mathbf{Q}^*\mathbf{D} \right] \\ &= \frac{2}{\Delta}\mathbf{A}_1\end{aligned}$$

where $\mathbf{D} = \left(\frac{2}{\Delta} - \boldsymbol{\Lambda}\right)^{-1}$ and \mathbf{A}_1 is defined as the term in the final brackets. Note that $(1 + \mathbf{Q}^*\mathbf{D}\mathbf{P})$ is actually a scalar in the case when the low-rank term has rank 1.

S4 Recurrence

Finally, the full bilinear discretization can be rewritten in terms of these matrices as

$$\begin{aligned}\overline{\mathbf{A}} &= \mathbf{A}_1\mathbf{A}_0 \\ \overline{\mathbf{B}} &= \frac{2}{\Delta}\mathbf{A}_1\Delta\mathbf{B} = 2\mathbf{A}_1\mathbf{B}.\end{aligned}$$

The discrete-time SSM (2.5) becomes

$$\begin{aligned} x_k &= \bar{\mathbf{A}}x_{k-1} + \bar{\mathbf{B}}u_k \\ &= \mathbf{A}_1\mathbf{A}_0x_{k-1} + 2\mathbf{A}_1\mathbf{B}u_k \\ y_k &= \mathbf{C}x_k. \end{aligned}$$

Note that $\mathbf{A}_0, \mathbf{A}_1$ are accessed only through matrix-vector multiplications (MVM). Since they are both DPLR, they have $O(N)$ MVM, showing Theorem 3.5.

C.1.2 Computing the Convolutional View

The most involved part of using SSMs efficiently is computing $\bar{\mathbf{K}}$. This algorithm was sketched in Section 3.3.1 and is the main motivation for the S4 parameterization. In this section, we define the necessary intermediate quantities and prove the main technical result.

The algorithm for Theorem 3.6 falls in roughly three stages, leading to Algorithm 1. Assuming \mathbf{A} has been conjugated into diagonal plus low-rank form, we successively simplify the problem of computing $\bar{\mathbf{K}}$ by applying the techniques outlined in Section 3.3.1.

We make an important note about a notational convenience for these proofs.

Remark C.1. *For the remainder of this section, we transpose \mathbf{C} to be a column vector of shape \mathbb{C}^N or $\mathbb{C}^{N \times 1}$ instead of matrix or row vector $\mathbb{C}^{1 \times N}$ as in (2.1). In other words the SSM is*

$$\begin{aligned} x'(t) &= \mathbf{A}x(t) + \mathbf{B}u(t) \\ y(t) &= \mathbf{C}^*x(t) + \mathbf{D}u(t). \end{aligned} \tag{C.1}$$

This convention is made so that \mathbf{C} has the same shape as $\mathbf{B}, \mathbf{P}, \mathbf{Q}$ and simplifies the implementation of S4.

DPLR Representation

Assume that \mathbf{A} is a (complex) diagonal plus low-rank (DPLR) matrix. For example, Chapter 6 is about how HIPPO matrices can be unitarily conjugated into DPLR matrices with Lemma 3.3.

Note that unlike diagonal matrices, a DPLR matrix does not lend itself to efficient computation of $\bar{\mathbf{K}}$. The reason is that $\bar{\mathbf{K}}$ computes terms $\mathbf{C}^*\bar{\mathbf{A}}^i\bar{\mathbf{B}}$ which involve powers of the

matrix $\bar{\mathbf{A}}$. These are trivially computable when $\bar{\mathbf{A}}$ is diagonal, but is no longer possible for even simple modifications to diagonal matrices such as DPLR.

Reduction 1: SSM Generating Function

To address the problem of computing powers of $\bar{\mathbf{A}}$, we introduce another technique. Instead of computing the SSM convolution filter $\bar{\mathbf{K}}$ directly, we introduce a generating function on its coefficients and compute evaluations of it.

Definition C.1 (SSM Generating Function). *We define the following quantities:*

- The SSM convolution function is $\mathcal{K}(\bar{\mathbf{A}}, \bar{\mathbf{B}}, \mathbf{C}) = (\mathbf{C}^* \bar{\mathbf{B}}, \mathbf{C}^* \bar{\mathbf{A}}\bar{\mathbf{B}}, \dots)$ and the (truncated) SSM filter of length L

$$\mathcal{K}_L(\bar{\mathbf{A}}, \bar{\mathbf{B}}, \mathbf{C}) = (\mathbf{C}^* \bar{\mathbf{B}}, \mathbf{C}^* \bar{\mathbf{A}}\bar{\mathbf{B}}, \dots, \mathbf{C}^* \bar{\mathbf{A}}^{L-1}\bar{\mathbf{B}}) \in \mathbb{R}^L \quad (\text{C.2})$$

- The SSM generating function at node z is

$$\hat{\mathcal{K}}(z; \bar{\mathbf{A}}, \bar{\mathbf{B}}, \mathbf{C}) \in \mathbb{C} := \sum_{i=0}^{\infty} \mathbf{C}^* \bar{\mathbf{A}}^i \bar{\mathbf{B}} z^i = \mathbf{C}^* (\mathbf{I} - \bar{\mathbf{A}}z)^{-1} \bar{\mathbf{B}} \quad (\text{C.3})$$

and the truncated SSM generating function at node z is

$$\hat{\mathcal{K}}_L(z; \bar{\mathbf{A}}, \bar{\mathbf{B}}, \mathbf{C})^* \in \mathbb{C} := \sum_{i=0}^{L-1} \mathbf{C}^* \bar{\mathbf{A}}^i \bar{\mathbf{B}} z^i = \mathbf{C}^* (\mathbf{I} - \bar{\mathbf{A}}^L z^L) (\mathbf{I} - \bar{\mathbf{A}}z)^{-1} \bar{\mathbf{B}} \quad (\text{C.4})$$

- The truncated SSM generating function at nodes $\Omega \in \mathbb{C}^M$ is

$$\hat{\mathcal{K}}_L(\Omega; \bar{\mathbf{A}}, \bar{\mathbf{B}}, \mathbf{C}) \in \mathbb{C}^M := \left(\hat{\mathcal{K}}_L(\omega_k; \bar{\mathbf{A}}, \bar{\mathbf{B}}, \mathbf{C}) \right)_{k \in [M]} \quad (\text{C.5})$$

Intuitively, the generating function essentially converts the SSM convolution filter from the time domain to frequency domain. Importantly, it preserves the same information, and the desired SSM convolution filter can be recovered from evaluations of its generating function.

Lemma C.2. *The SSM function $\mathcal{K}_L(\bar{\mathbf{A}}, \bar{\mathbf{B}}, \mathbf{C})$ can be computed from the SSM generating function $\hat{\mathcal{K}}_L(\Omega; \bar{\mathbf{A}}, \bar{\mathbf{B}}, \mathbf{C})$ at the roots of unity $\Omega = \{\exp(-2\pi i \frac{k}{L}) : k \in [L]\}$ stably in $O(L \log L)$ operations.*

Proof. For convenience define

$$\begin{aligned}\overline{\mathbf{K}} &= \mathcal{K}_L(\overline{\mathbf{A}}, \overline{\mathbf{B}}, \mathbf{C}) \\ \hat{\mathbf{K}} &= \hat{\mathcal{K}}_L(\Omega; \overline{\mathbf{A}}, \overline{\mathbf{B}}, \mathbf{C}) \\ \hat{\mathbf{K}}(z) &= \hat{\mathcal{K}}_L(z; \overline{\mathbf{A}}, \overline{\mathbf{B}}, \mathbf{C}).\end{aligned}$$

Note that

$$\hat{\mathbf{K}}_j = \sum_{k=0}^{L-1} \overline{\mathbf{K}}_k \exp\left(-2\pi i \frac{jk}{L}\right).$$

Note that this is exactly the same as the Discrete Fourier Transform (DFT):

$$\hat{\mathbf{K}} = \mathcal{F}_L \mathbf{K}.$$

Therefore \mathbf{K} can be recovered from $\hat{\mathbf{K}}$ with a single inverse DFT, which requires $O(L \log L)$ operations with the Fast Fourier Transform (FFT) algorithm. \square

Reduction 2: Woodbury Correction The primary motivation of Definition C.1 is that it turns *powers* of $\overline{\mathbf{A}}$ into a single *inverse* of $\overline{\mathbf{A}}$ (equation (C.3)). While DPLR matrices cannot be powered efficiently due to the low-rank term, they can be inverted efficiently by the well-known Woodbury identity (Proposition A.2).

With this identity, we can convert the SSM generating function on a DPLR matrix \mathbf{A} into one on just its diagonal component.

Lemma C.3. *Let $\mathbf{A} = \boldsymbol{\Lambda} - \mathbf{P}\mathbf{Q}^*$ be a diagonal plus low-rank representation. Then for any root of unity $z \in \Omega$, the truncated generating function satisfies*

$$\begin{aligned}\hat{\mathbf{K}}(z) &= \frac{2}{1+z} \left[\tilde{\mathbf{C}}^* \mathbf{R}(z) \mathbf{B} - \tilde{\mathbf{C}}^* \mathbf{R}(z) \mathbf{P} (1 + \mathbf{Q}^* \mathbf{R}(z) \mathbf{P})^{-1} \mathbf{Q}^* \mathbf{R}(z) \mathbf{B} \right] \\ \tilde{\mathbf{C}} &= (\mathbf{I} - \overline{\mathbf{A}}^L)^* \mathbf{C} \\ \mathbf{R}(z; \boldsymbol{\Lambda}) &= \left(\frac{2}{\Delta} \frac{1-z}{1+z} - \boldsymbol{\Lambda} \right)^{-1}.\end{aligned}$$

Proof. Directly expanding Definition C.1 yields

$$\begin{aligned}\mathcal{K}_L(z; \bar{\mathbf{A}}, \bar{\mathbf{B}}, \mathbf{C}) &= \mathbf{C}^* \bar{\mathbf{B}} + \mathbf{C}^* \bar{\mathbf{A}} \bar{\mathbf{B}} z + \cdots + \mathbf{C}^* \bar{\mathbf{A}}^{L-1} \bar{\mathbf{B}} z^{L-1} \\ &= \mathbf{C}^* \left(\mathbf{I} - \bar{\mathbf{A}}^L \right) \left(\mathbf{I} - \bar{\mathbf{A}} z \right)^{-1} \bar{\mathbf{B}} \\ &= \tilde{\mathbf{C}}^* \left(\mathbf{I} - \bar{\mathbf{A}} z \right)^{-1} \bar{\mathbf{B}}\end{aligned}$$

where $\tilde{\mathbf{C}}^* = \mathbf{C}^* \left(\mathbf{I} - \bar{\mathbf{A}}^L \right)$.

We can now explicitly expand the discretized SSM matrices $\bar{\mathbf{A}}$ and $\bar{\mathbf{B}}$ back in terms of the original SSM parameters \mathbf{A} and \mathbf{B} . Lemma C.4 provides an explicit formula, which allows further simplifying

$$\begin{aligned}\tilde{\mathbf{C}}^* \left(\mathbf{I} - \bar{\mathbf{A}} z \right)^{-1} \bar{\mathbf{B}} &= \frac{2}{1+z} \tilde{\mathbf{C}}^* \left(\frac{2}{\Delta} \frac{1-z}{1+z} - \mathbf{A} \right)^{-1} \mathbf{B} \\ &= \frac{2}{1+z} \tilde{\mathbf{C}}^* \left(\frac{2}{\Delta} \frac{1-z}{1+z} - \mathbf{\Lambda} + \mathbf{P} \mathbf{Q}^* \right)^{-1} \mathbf{B} \\ &= \frac{2}{1+z} \left[\tilde{\mathbf{C}}^* \mathbf{R}(z) \mathbf{B} - \tilde{\mathbf{C}}^* \mathbf{R}(z) \mathbf{P} (1 + \mathbf{Q}^* \mathbf{R}(z) \mathbf{P})^{-1} \mathbf{Q}^* \mathbf{R}(z) \mathbf{B} \right].\end{aligned}$$

The last line applies the Woodbury Identity (Proposition A.2) where $\mathbf{R}(z) = \left(\frac{2}{\Delta} \frac{1-z}{1+z} - \mathbf{\Lambda} \right)^{-1}$. \square

The previous proof used the following self-contained result to back out the original SSM matrices from the discretization.

Lemma C.4. *Let $\bar{\mathbf{A}}, \bar{\mathbf{B}}$ be the SSM matrices \mathbf{A}, \mathbf{B} discretized by the bilinear discretization with step size Δ . Then*

$$\mathbf{C}^* \left(\mathbf{I} - \bar{\mathbf{A}} z \right)^{-1} \bar{\mathbf{B}} = \frac{2\Delta}{1+z} \mathbf{C}^* \left[2 \frac{1-z}{1+z} - \Delta \mathbf{A} \right]^{-1} \mathbf{B}$$

Proof. Recall that the bilinear discretization that we use (equation (2.5)) is

$$\begin{aligned}\bar{\mathbf{A}} &= \left(\mathbf{I} - \frac{\Delta}{2} \mathbf{A} \right)^{-1} \left(\mathbf{I} + \frac{\Delta}{2} \mathbf{A} \right) \\ \bar{\mathbf{B}} &= \left(\mathbf{I} - \frac{\Delta}{2} \mathbf{A} \right)^{-1} \Delta \mathbf{B}\end{aligned}$$

The result is proved algebraic manipulations.

$$\begin{aligned}
\mathbf{C}^* (\mathbf{I} - \overline{\mathbf{A}}z)^{-1} \overline{\mathbf{B}} &= \mathbf{C}^* \left[\left(\mathbf{I} - \frac{\Delta}{2} \mathbf{A} \right)^{-1} \left(\mathbf{I} - \frac{\Delta}{2} \mathbf{A} \right) - \left(\mathbf{I} - \frac{\Delta}{2} \mathbf{A} \right)^{-1} \left(\mathbf{I} + \frac{\Delta}{2} \mathbf{A} \right) z \right]^{-1} \overline{\mathbf{B}} \\
&= \mathbf{C}^* \left[\left(\mathbf{I} - \frac{\Delta}{2} \mathbf{A} \right) - \left(\mathbf{I} + \frac{\Delta}{2} \mathbf{A} \right) z \right]^{-1} \left(\mathbf{I} - \frac{\Delta}{2} \mathbf{A} \right) \overline{\mathbf{B}} \\
&= \mathbf{C}^* \left[\mathbf{I}(1-z) - \frac{\Delta}{2} \mathbf{A}(1+z) \right]^{-1} \Delta \mathbf{B} \\
&= \frac{\Delta}{1-z} \mathbf{C}^* \left[\mathbf{I} - \frac{\Delta \mathbf{A}}{2 \frac{1-z}{1+z}} \right]^{-1} \mathbf{B} \\
&= \frac{2\Delta}{1+z} \mathbf{C}^* \left[2 \frac{1-z}{1+z} \mathbf{I} - \Delta \mathbf{A} \right]^{-1} \mathbf{B}
\end{aligned}$$

□

Note that in the S4 parameterization, instead of constantly computing $\tilde{\mathbf{C}} = (\mathbf{I} - \overline{\mathbf{A}}^L)^* \mathbf{C}$, we can simply reparameterize our parameters to learn $\tilde{\mathbf{C}}$ directly instead of \mathbf{C} , saving a minor computation cost and simplifying the algorithm.

Reduction 3: Cauchy Kernel We have reduced the original problem of computing $\overline{\mathbf{K}}$ to the problem of computing the SSM generating function $\hat{\mathcal{K}}_L(\Omega; \overline{\mathbf{A}}, \overline{\mathbf{B}}, \mathbf{C})$ in the case that $\overline{\mathbf{A}}$ is a diagonal matrix. We show that this is exactly the same as a Cauchy kernel, which is a well-studied problem with fast and stable numerical algorithms.

Definition C.5. A *Cauchy matrix* or *kernel* on nodes $\Omega = (\omega_i) \in \mathbb{C}^M$ and $\Lambda = (\lambda_j) \in \mathbb{C}^N$ is

$$\mathbf{M} \in \mathbb{C}^{M \times N} = \mathbf{M}(\Omega, \Lambda) = (\mathbf{M}_{ij})_{i \in [M], j \in [N]} \quad \mathbf{M}_{ij} = \frac{1}{\omega_i - \lambda_j}.$$

The computation time of a Cauchy matrix-vector multiplication of size $M \times N$ is denoted by $\mathcal{C}(M, N)$.

Computing with Cauchy matrices is an extremely well-studied problem in numerical analysis, with both fast arithmetic algorithms and fast numerical algorithms based on the famous Fast Multipole Method (FMM) [152, 154, 153].

Proposition C.6 (Cauchy). A Cauchy kernel requires $O(M + N)$ space, and operation

count

$$\mathcal{C}(M, N) = \begin{cases} O(MN) & \text{naively} \\ O((M+N)\log^2(M+N)) & \text{in exact arithmetic} \\ O((M+N)\log(M+N)\log\frac{1}{\varepsilon}) & \text{numerically to precision } \varepsilon. \end{cases}$$

Corollary C.7. *Evaluating $\mathbf{Q}^* \mathbf{R}(\Omega; \Lambda) \mathbf{P}$ (defined in Lemma C.3) for any set of nodes $\Omega \in \mathbb{C}^L$, diagonal matrix Λ , and vectors \mathbf{P}, \mathbf{Q} can be computed in $\mathcal{C}(L, N)$ operations and $O(L + N)$ space, where $\mathcal{C}(L, N) = \tilde{O}(L + N)$ is the cost of a Cauchy matrix-vector multiplication.*

Proof. For any fixed $\omega \in \Omega$, we want to compute $\sum_j \frac{q_j^* p_j}{\omega - \lambda_j}$. Computing this over all ω_i is therefore exactly a Cauchy matrix-vector multiplication. \square

This completes the proof of Theorem 3.6. In Algorithm 1, note that the work is dominated by Step 2, which has a constant number of calls to a black-box Cauchy kernel, with complexity given by Proposition C.6.

C.2 Computation Complexity Comparison: SSM, S4, and Other Sequence Models

We provide a summary of complexity requirements for various sequence model mechanisms, including several versions of S4. This elaborates on Table 3.2.

First, we decouple the model dimension H and the state dimension N for finer-grained complexities. Second, we include additional variants of S4. In Table C.1, SSM-naive refers to an SSM with general unstructured (dense) \mathbf{A} . S4-fixed denotes not training $(\Delta, \mathbf{A}, \mathbf{B})$ and computing with the simple algorithm for frozen matrices discussed in Section 3.1.2. S4 denotes training all parameters with the full S4 algorithm (either DPLR or Diagonal).

We also include brief explanations of these complexities for the SSM variants.

SSM-naive

- Parameters: $O(HN)$ in the matrices \mathbf{B}, \mathbf{C} and $O(N^2)$ in the matrix \mathbf{A} .

Table C.1: (**Sequence model complexities (extended).**) Complexity of various sequence models in terms of length (L), batch size (B), and hidden dimension (H). Metrics are parameter count, training computation, space requirement, and inference computation for 1 sample and time-step.

	Convolution	RNN
Parameters	LH^2	H^2
Training	$BLH^2 + L \log(L)(H^2 + BH)$	BLH^2
Space	$BLH + LH^2$	BLH
Parallel	Yes	No
Inference	LH^2	H^2
	Attention	SSM-naive
Parameters	H^2	$HN + N^2$
Training	$B(L^2H + LH^2)$	$HN^3 + LHN^2 + BL \log(L)HN$
Space	$B(L^2 + HL)$	$HN^2 + LHN + BLH$
Parallel	Yes	Yes
Inference	$L^2H + H^2L$	HN^2
	S4-fixed	S4
Parameters	HN	HN
Training	$BL \log(L)HN$	$BH(N \log^2 N + L \log L) + BL \log(L)H$
Space	$LHN + BLH$	BHL
Parallel	Yes	Yes
Inference	HN^2	HN

- Training: $O(HN^3)$ to invert compute the matrix $\bar{\mathbf{A}}$ for all H features. $O(LHN^2)$ to compute the SSK $\mathbf{C}, \mathbf{CA}, \dots$. $O(BL \log(L)HN)$ to multiply by \mathbf{B} and convolve with u .
- Space $O(HN^2)$ to store $\bar{\mathbf{A}}$. $O(LHN)$ to store the SSM kernel. $O(BLH)$ to store the inputs and outputs.
- Inference: $O(HN^2)$ for MVM by $\bar{\mathbf{A}}$.

S4-fixed

- Parameters: $O(HN)$ in the matrices \mathbf{C} .
- Training: $O(BL \log(L)H)$ to convolve with u .
- Space $O(LHN)$ to store the SSM kernel. $O(BLH)$ for inputs/outputs.
- Inference: $O(HN^2)$ for MVM by $\bar{\mathbf{A}}$.

S4

- Parameters: $O(HN)$ for $\Delta, \mathbf{A}, \mathbf{B}, \mathbf{C}$.
- Training: $BH \cdot \tilde{O}(N + L)$ to compute the SSK, $O(BL \log(L)H)$ for the convolution.
- Space $O(BHL)$ to store the SSK (and inputs/outputs).
- Inference: $O(HN)$ to multiply x_t (shape $[H, N]$) by $\bar{\mathbf{A}}$ (shape $[H, N, N]$).

Appendix D

Appendix for Chapter 4

D.1 General HIPPO Framework

We present the general HIPPO framework, as described in Section 4.2, in more details. We also generalize it to include bases other than polynomials.

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 $f: \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$, HIPPO defines an operator that maps f to the optimal projection coefficients $c: \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^N$, such that

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

The first step refers to the `projt` operator and the second the `coeft` operator in Definition 4.1.

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

We first describe the parameters of the `hippo` operator (a measure and basis) in more detail in Appendix D.1.1. We define the projection `projt` and coefficient `coeft` operators in Appendix D.1.2. Then we give a general strategy to calculate these coefficients $c(t)$, by deriving a differential equation that governs the coefficient dynamics (Appendix D.1.3). Finally we discuss how to turn the continuous `hippo` operator into a discrete one that can be applied to sequence data (Appendix D.1.4).

D.1.1 Measure and Basis

We describe and motivate the ingredients of HIPPO in more detail here. Recall that the high level goal is online function approximation; this requires both a set of valid approximations and a notion of approximation quality.

Approximation Measures

At every t , the approximation quality is defined with respect to a measure $\mu^{(t)}$ supported on $(-\infty, t]$. We seek some polynomial $g^{(t)}$ of degree at most $N - 1$ that minimizes the error $\|f_{x \leq t} - g^{(t)}\|_{L_2(\mu^{(t)})}$. Intuitively, this measure $\mu^{(t)}$ governs how much to weigh every time in the past. For simplicity, we assume that the measures $\mu^{(t)}$ are sufficiently smooth across their domain as well as in time; in particular, they have densities $\omega(t, x) := \frac{d\mu^{(t)}}{d\lambda}(x)$ with respect to the Lebesgue measure $d\lambda(x) := dx$ such that ω is C^1 almost everywhere. Thus integrating against $d\mu^{(t)}(x)$ can be rewritten as integrating against $\omega(t, x) dx$.

We also assume for simplicity that the measures $\mu^{(t)}$ are normalized to be probability measures; arbitrary scaling does not affect the optimal projection.

Orthogonal Polynomial Basis

Let $\{P_n\}_{n \in \mathbb{N}}$ denote a sequence of orthogonal polynomials with respect to some base measure μ . Similarly define $\{P_n^{(t)}\}_{n \in \mathbb{N}}$ to be a sequence of orthogonal polynomials with respect to the time-varying measure $\mu^{(t)}$. Let $p_n^{(t)}$ be the normalized version of $P_n^{(t)}$ (i.e., have norm 1 w.r.t. $\mu^{(t)}$), and define

$$p_n(t, x) = p_n^{(t)}(x). \quad (\text{D.1})$$

Note that the $P_n^{(t)}$ are not required to be normalized, while the $p_n^{(t)}$ are.

Tilted Measure and Basis

Our goal is simply to store a compressed representation of functions, which can use any basis, not necessarily OPs. For any scaling function

$$\chi(t, x) = \chi^{(t)}(x), \quad (\text{D.2})$$

the functions $p_n^{(t)}(x)\chi^{(t)}(x)$ are orthogonal with respect to the density ω/χ^2 at every time t . Thus, we can choose this alternative basis and measure to perform the projections.

To formalize this tilting with χ , define $\nu^{(t)}$ to be the normalized measure with density

proportional to $\omega^{(t)} / (\chi^{(t)})^2$. We will calculate the normalized measure and the orthonormal basis for it. Let

$$\zeta(t) = \int \frac{\omega}{\chi^2} = \int \frac{\omega^{(t)}(x)}{(\chi^{(t)}(x))^2} dx \quad (\text{D.3})$$

be the normalization constant, so that $\nu^{(t)}$ has density $\frac{\omega^{(t)}}{\zeta(t)(\chi^{(t)})^2}$. If $\chi(t, x) = 1$ (no tilting), this constant is $\zeta(t) = 1$. In general, we assume that ζ is constant for all t ; if not, it can be folded into χ directly.

Next, note that (dropping the dependence on x inside the integral for shorthand)

$$\begin{aligned} \left\| \zeta(t)^{\frac{1}{2}} p_n^{(t)} \chi^{(t)} \right\|_{\nu^{(t)}}^2 &= \int \left(\zeta(t)^{\frac{1}{2}} p_n^{(t)} \chi^{(t)} \right)^2 \frac{\omega^{(t)}}{\zeta(t)(\chi^{(t)})^2} \\ &= \int (p_n^{(t)})^2 \omega^{(t)} \\ &= \left\| p_n^{(t)} \right\|_{\mu^{(t)}}^2 = 1. \end{aligned}$$

Thus we define the orthogonal basis for $\nu^{(t)}$

$$g_n^{(t)} = \lambda_n \zeta(t)^{\frac{1}{2}} p_n^{(t)} \chi^{(t)}, \quad n \in \mathbb{N}. \quad (\text{D.4})$$

We let each element of the basis be scaled by a λ_n scalar, for reasons discussed soon, since arbitrary scaling does not change orthogonality:

$$\langle g_n^{(t)}, g_m^{(t)} \rangle_{\nu^{(t)}} = \lambda_n^2 \delta_{n,m}$$

Note that when $\lambda_n = \pm 1$, the basis $\{g_n^{(t)}\}$ is an orthonormal basis with respect to the measure $\nu^{(t)}$, at every time t . Notationally, let $g_n(t, x) := g_n^{(t)}(x)$ as usual.

Note that in the case $\chi = 1$ (i.e., no tilting), we also have $\zeta = 1$ and $g_n = \lambda_n p_n$ (for all t, x).

Remark D.1. *The tilting χ is not used in the derivations of the main methods in this section, LegT and LegS (equivalently, assume $\chi = 1$). We include it because the original the HIPPO framework included a third method called LagT which required it; additionally, it is needed in the extension of HIPPO in Chapter 5, so we introduce the concept now.*

D.1.2 The Projection and Coefficients

Given a choice of measures and basis functions, we next see how the coefficients $c(t)$ can be computed.

Input: Function We are given a C^1 -smooth function $f : [0, \infty) \rightarrow \mathbb{R}$ which is seen *online*, for which we wish to maintain a compressed representation of its history $f(x)_{\leq t} = f(x)_{x \leq t}$ at every time t .

Output: Approximation Coefficients The function f can be approximated by storing its coefficients with respect to the basis $\{g_n\}_{n < N}$. For example, in the case of no tilting $\chi = 1$, this encodes the optimal polynomial approximation of f of degree less than N . In particular, at time t we wish to represent $f_{\leq t}$ as a linear combination of polynomials $g_n^{(t)}$. Since the $g_n^{(t)}$ are orthogonal with respect to the Hilbert space defined by $\langle \cdot, \cdot \rangle_{\nu^{(t)}}$, it suffices to calculate coefficients

$$\begin{aligned} c_n(t) &= \langle f_{\leq t}, g_n^{(t)} \rangle_{\nu^{(t)}} \\ &= \int f g_n^{(t)} \frac{\omega^{(t)}}{\zeta(t)(\chi^{(t)})^2} \\ &= \zeta(t)^{-\frac{1}{2}} \lambda_n \int f p_n^{(t)} \frac{\omega^{(t)}}{\chi^{(t)}}. \end{aligned} \tag{D.5}$$

Reconstruction At any time t , $f_{\leq t}$ can be explicitly reconstructed as

$$\begin{aligned} f_{\leq t} \approx g^{(t)} &:= \sum_{n=0}^{N-1} \langle f_{\leq t}, g_n^{(t)} \rangle_{\nu^{(t)}} \frac{g_n^{(t)}}{\|g_n^{(t)}\|_{\nu^{(t)}}^2} \\ &= \sum_{n=0}^{N-1} \lambda_n^{-2} c_n(t) g_n^{(t)} \\ &= \sum_{n=0}^{N-1} \lambda_n^{-1} \zeta^{\frac{1}{2}} c_n(t) p_n^{(t)} \chi^{(t)}. \end{aligned} \tag{D.6}$$

Equation (D.6) is the proj_t operator; given the measure and basis parameters, it defines the optimal approximation of $f_{\leq t}$.

The coef_t operator simply extracts the vector of coefficients $c(t) = (c_n(t))_{n \in [N]}$.

D.1.3 Coefficient Dynamics: the HIPPO Operator

For the purposes of end-to-end models consuming an input function $f(t)$, the coefficients $c(t)$ are enough to encode information about the history of f and allow online predictions. Therefore, defining $c(t)$ to be the vector of $c_n(t)$ from equation (D.5), our focus will be on how to calculate the function $c : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^N$ from the input function $f : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$.

In our framework, we will compute these coefficients over time by viewing them as a dynamical system. Differentiating (D.5),

$$\begin{aligned} \frac{d}{dt}c_n(t) &= \zeta(t)^{-\frac{1}{2}}\lambda_n \int f(x) \left(\frac{\partial}{\partial t} p_n(t, x) \right) \frac{\omega}{\chi}(t, x) dx \\ &\quad + \int f(x) \left(\zeta^{-\frac{1}{2}}\lambda_n p_n(t, x) \right) \left(\frac{\partial}{\partial t} \frac{\omega}{\chi}(t, x) \right) dx. \end{aligned} \tag{D.7}$$

Here we have made use of the assumption that ζ is constant for all t .

Let $c(t) \in \mathbb{R}^N$ denote the vector of all coefficients $(c_n(t))_{0 \leq n < N}$.

The key idea is that if $\frac{\partial}{\partial t} P_n$ and $\frac{\partial}{\partial t} \frac{\omega}{\chi}$ have closed forms that can be related back to the polynomials P_k , then an ordinary differential equation can be written for $c(t)$. This allows these coefficients $c(t)$ and hence the optimal polynomial approximation to be computed online. Since $\frac{d}{dt} P_n^{(t)}$ is a polynomial (in x) of degree $n - 1$, it can be written as linear combinations of P_0, \dots, P_{n-1} , so the first term in Equation (D.7) is a linear combination of c_0, \dots, c_{n-1} . For many weight functions ω , we can find scaling function χ such that $\frac{\partial}{\partial t} \frac{\omega}{\chi}$ can also be written in terms of $\frac{\omega}{\chi}$ itself, and thus in those cases the second term of (D.7) is also a linear combination of c_0, \dots, c_{N-1} and the input f . Thus this often yields a closed-form linear ODE for $c(t)$.

Normalized dynamics Our purpose of defining the free parameters λ_n was threefold.

1. First, note that the orthonormal basis is not unique, up to a ± 1 factor per element.
2. Second, choosing λ_n can help simplify the derivations.
3. Third, although choosing $\lambda_n = \pm 1$ will be our default, since projecting onto an orthonormal basis is most sensible, the LMU [218] used a different scaling. Appendix D.2.1 will recover the LMU by choosing different λ_n for the LegT measure.

Suppose that equation (D.7) reduced to dynamics of the form

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

Then, letting $\Lambda = \text{diag}_{n \in [N]} \{\lambda_n\}$,

$$\frac{d}{dt}\Lambda^{-1}c(t) = -\Lambda^{-1}\mathbf{A}(t)\Lambda\Lambda^{-1}c(t) + \Lambda^{-1}\mathbf{B}(t)f(t).$$

Therefore, if we reparameterize the coefficients ($\Lambda^{-1}c(t) \rightarrow c(t)$) then the *normalized* coefficients projected onto the orthonormal basis satisfy dynamics and associated reconstruction

$$\frac{d}{dt}c(t) = -(\Lambda^{-1}\mathbf{A}(t)\Lambda)c(t) + (\Lambda^{-1}\mathbf{B}(t))f(t) \quad (\text{D.8})$$

$$f_{\leq t} \approx g^{(t)} = \sum_{n=0}^{N-1} \zeta^{\frac{1}{2}} c_n(t) p_n^{(t)} \chi^{(t)} \quad (\text{D.9})$$

These are the `hippo` and `projt` operators.

D.1.4 Discretization

As defined here, `hippo` is a map on continuous functions. However, as `hippo` defines a closed-form ODE of the coefficient dynamics, standard ODE discretization methods (Appendix A.1) can be applied to turn this into discrete memory updates. Thus we overload these operators, i.e. `hippo` either defines an ODE of the form

$$c'(t) = \mathbf{A}(t)c(t) + \mathbf{B}(t)f(t)$$

or a recurrence

$$c_k = \mathbf{A}_k c_{k-1} + \mathbf{B}_k f_k,$$

whichever is clear from context.

D.2 Derivations of HIPPO Projection Operators

We derive the memory updates associated with the translated Legendre (LegT) measure (Section 4.2.3), along with the scaled Legendre (LegS) measure (Section 4.3).

The majority of the work has already been accomplished by setting up the projection framework, and the proof simply requires following the technical outline laid out in Appendix D.1. In particular, the definition of the coefficients (D.5) and reconstruction (D.6) does not change, and we only consider how to calculate the coefficients dynamics (D.7).

For each case, we follow the general steps:

Measure and Basis: Define the measure $\mu^{(t)}$ or weight $\omega(t, x)$ and basis functions $p_n(t, x)$.

Derivatives: Compute the derivatives of the measure and basis functions,

Coefficient Dynamics: Plug them into the coefficient dynamics (D.7) to derive the ODE that describes how to compute the coefficients $c(t)$.

Reconstruction: Provide the complete formula to reconstruct an approximation to the function $f_{\leq t}$, which is the optimal projection under this measure and basis.

The derivation in Appendix D.2.1 proves Theorem 4.2, and the derivation in Appendix D.2.2 proves Theorem 4.3.

D.2.1 Derivation for Translated Legendre (HIPPO-LegT)

This measure fixes a window length θ and slides it across time.

Measure and Basis

We use a uniform weight function supported on the interval $[t - \theta, t]$ and pick Legendre polynomials $P_n(x)$, translated from $[-1, 1]$ to $[t - \theta, t]$, as basis functions:

$$\begin{aligned}\omega(t, x) &= \frac{1}{\theta} \mathbb{I}_{[t-\theta, t]} \\ p_n(t, x) &= (2n+1)^{1/2} P_n\left(\frac{2(x-t)}{\theta} + 1\right) \\ g_n(t, x) &= \lambda_n p_n(t, x).\end{aligned}$$

Here, we have used no tilting so $\chi = 1$ and $\zeta = 1$ (equations (D.2) and (D.3)). We leave λ_n unspecified for now.

At the endpoints, these basis functions satisfy

$$\begin{aligned} g_n(t, t) &= \lambda_n(2n+1)^{\frac{1}{2}} \\ g_n(t, t-\theta) &= \lambda_n(-1)^n(2n+1)^{\frac{1}{2}}. \end{aligned}$$

Derivatives

The derivative of the measure is

$$\frac{\partial}{\partial t}\omega(t, x) = \frac{1}{\theta}\delta_t - \frac{1}{\theta}\delta_{t-\theta}.$$

The derivative of Legendre polynomials can be expressed as linear combinations of other Legendre polynomials (cf. Appendix A.2.1).

$$\begin{aligned} \frac{\partial}{\partial t}g_n(t, x) &= \lambda_n(2n+1)^{\frac{1}{2}} \cdot \frac{-2}{\theta}P'_n\left(\frac{2(x-t)}{\theta} + 1\right) \\ &= \lambda_n(2n+1)^{\frac{1}{2}}\frac{-2}{\theta}\left[(2n-1)P_{n-1}\left(\frac{2(x-t)}{\theta} + 1\right)\right. \\ &\quad \left.+ (2n-5)P_{n-3}\left(\frac{2(x-t)}{\theta} + 1\right) + \dots\right] \\ &= -\lambda_n(2n+1)^{\frac{1}{2}}\frac{2}{\theta}\left[\lambda_{n-1}^{-1}(2n-1)^{\frac{1}{2}}g_{n-1}(t, x) + \lambda_{n-3}^{-1}(2n-3)^{\frac{1}{2}}g_{n-3}(t, x) + \dots\right]. \end{aligned}$$

We have used equation (A.8) here.

Sliding Approximation

As a special case for the LegT measure, we need to consider an approximation due to the nature of the sliding window measure.

When analyzing $\frac{d}{dt}c(t)$ in the next section, we will need to use the value $f(t-\theta)$. However, at time t this input is no longer available. Instead, we need to rely on our compressed representation of the function: by the reconstruction equation (D.6), if the approximation

is succeeding so far, we should have

$$\begin{aligned} f_{\leq t}(x) &\approx \sum_{k=0}^{N-1} \lambda_k^{-1} c_k(t) (2k+1)^{\frac{1}{2}} P_k \left(\frac{2(x-t)}{\theta} + 1 \right) \\ f(t-\theta) &\approx \sum_{k=0}^{N-1} \lambda_k^{-1} c_k(t) (2k+1)^{\frac{1}{2}} (-1)^k. \end{aligned}$$

Coefficient Dynamics

We are ready to derive the coefficient dynamics.

Plugging the derivatives of this measure and basis into equation (D.7) gives

$$\begin{aligned} \frac{d}{dt} c_n(t) &= \int f(x) \left(\frac{\partial}{\partial t} g_n(t, x) \right) \omega(t, x) dx \\ &\quad + \int f(x) g_n(t, x) \left(\frac{\partial}{\partial t} \omega(t, x) \right) dx \\ &= -\lambda_n (2n+1)^{\frac{1}{2}} \frac{2}{\theta} \left[\lambda_{n-1}^{-1} (2n-1)^{\frac{1}{2}} c_{n-1}(t) + \lambda_{n-3}^{-1} (2n-5)^{\frac{1}{2}} c_{n-3}(t) + \dots \right] \\ &\quad + \frac{1}{\theta} f(t) g_n(t, t) - \frac{1}{\theta} f(t-\theta) g_n(t, t-\theta) \\ &\approx -\frac{\lambda_n}{\theta} (2n+1)^{\frac{1}{2}} \cdot 2 \left[(2n-1)^{\frac{1}{2}} \frac{c_{n-1}(t)}{\lambda_{n-1}} + (2n-5)^{\frac{1}{2}} \frac{c_{n-3}(t)}{\lambda_{n-3}} + \dots \right] \\ &\quad + (2n+1)^{\frac{1}{2}} \frac{\lambda_n}{\theta} f(t) - (2n+1)^{\frac{1}{2}} \frac{\lambda_n}{\theta} (-1)^n \sum_{k=0}^{N-1} (2k+1)^{\frac{1}{2}} \frac{c_k(t)}{\lambda_k} (-1)^k \\ &= -\frac{\lambda_n}{\theta} (2n+1)^{\frac{1}{2}} \cdot 2 \left[(2n-1)^{\frac{1}{2}} \frac{c_{n-1}(t)}{\lambda_{n-1}} + (2n-5)^{\frac{1}{2}} \frac{c_{n-3}(t)}{\lambda_{n-3}} + \dots \right] \\ &\quad - (2n+1)^{\frac{1}{2}} \frac{\lambda_n}{\theta} \sum_{k=0}^{N-1} (-1)^{n-k} (2k+1)^{\frac{1}{2}} \frac{c_k(t)}{\lambda_k} + (2n+1)^{\frac{1}{2}} \frac{\lambda_n}{\theta} f(t) \\ &= -\frac{\lambda_n}{\theta} (2n+1)^{\frac{1}{2}} \sum_{k=0}^{N-1} M_{nk} (2k+1)^{\frac{1}{2}} \frac{c_k(t)}{\lambda_k} + (2n+1)^{\frac{1}{2}} \frac{\lambda_n}{\theta} f(t), \end{aligned}$$

where

$$M_{nk} = \begin{cases} 1 & \text{if } k \leq n \\ (-1)^{n-k} & \text{if } k \geq n \end{cases}.$$

Now we consider two instantiations for λ_n . The first one is the more natural $\lambda_n = 1$, which turns g_n into an orthonormal basis. We then get

$$\begin{aligned} c'(t) &= -\frac{1}{\theta} \mathbf{A}c(t) + \frac{1}{\theta} \mathbf{B}f(t) \\ \mathbf{A}_{nk} &= (2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} \begin{cases} 1 & \text{if } k \leq n \\ (-1)^{n-k} & \text{if } k \geq n \end{cases} \\ \mathbf{B}_n &= (2n+1)^{\frac{1}{2}}. \end{aligned}$$

The second case takes $\lambda_n = (2n+1)^{\frac{1}{2}}(-1)^n$. This yields

$$\begin{aligned} c'(t) &= -\frac{1}{\theta} \mathbf{A}c(t) + \frac{1}{\theta} \mathbf{B}f(t) \\ \mathbf{A}_{nk} &= (2n+1) \begin{cases} (-1)^{n-k} & \text{if } k \leq n \\ 1 & \text{if } k \geq n \end{cases} \\ \mathbf{B}_n &= (2n+1)(-1)^n. \end{aligned}$$

This is exactly the LMU update equation.

Reconstruction

By equation (D.6), at every time t we have

$$f(x) \approx g^{(t)}(x) = \sum_n \lambda_n^{-1} c_n(t) (2n+1)^{\frac{1}{2}} P_n \left(\frac{2(x-t)}{\theta} + 1 \right).$$

D.2.2 Derivation for Scaled Legendre (HIPPO-LegS)

Out of all the methods discussed in Chapters 4 and 5, the scaled Legendre is our only method that uses a measure with varying width.

Measure and Basis

We instantiate the framework in the case

$$\omega(t, x) = \frac{1}{t} \mathbb{I}_{[0,t]} \tag{D.10}$$

$$g_n(t, x) = p_n(t, x) = (2n+1)^{\frac{1}{2}} P_n \left(\frac{2x}{t} - 1 \right) \tag{D.11}$$

Here, P_n are the basic Legendre polynomials (Appendix A.2.1). We use no tilting, i.e. $\chi(t, x) = 1$, $\zeta(t) = 1$, and $\lambda_n = 1$ so that the functions $g_n(t, x)$ are an orthonormal basis.

Derivatives

We first differentiate the measure and basis:

$$\begin{aligned}\frac{\partial}{\partial t} \omega(t, \cdot) &= -t^{-2} \mathbb{I}_{[0,t]} + t^{-1} \delta_t = t^{-1}(-\omega(t) + \delta_t) \\ \frac{\partial}{\partial t} g_n(t, x) &= -(2n+1)^{\frac{1}{2}} 2xt^{-2} P'_n\left(\frac{2x}{t} - 1\right) \\ &= -(2n+1)^{\frac{1}{2}} t^{-1} \left(\frac{2x}{t} - 1 + 1\right) P'_n\left(\frac{2x}{t} - 1\right).\end{aligned}$$

Now define $z = \frac{2x}{t} - 1$ for shorthand and apply the properties of derivatives of Legendre polynomials (equation (A.9)).

$$\begin{aligned}\frac{\partial}{\partial t} g_n(t, x) &= -(2n+1)^{\frac{1}{2}} t^{-1} (z+1) P'_n(z) \\ &= -(2n+1)^{\frac{1}{2}} t^{-1} [nP_n(z) + (2n-1)P_{n-1}(z) + (2n-3)P_{n-2}(z) + \dots] \\ &= -t^{-1} (2n+1)^{\frac{1}{2}} \left[n(2n+1)^{-\frac{1}{2}} g_n(t, x) + (2n-1)^{\frac{1}{2}} g_{n-1}(t, x) + (2n-3)^{\frac{1}{2}} g_{n-2}(t, x) + \dots \right]\end{aligned}$$

Coefficient Dynamics

Plugging these into (D.7), we obtain

$$\begin{aligned}\frac{d}{dt} c_n(t) &= \int f(x) \left(\frac{\partial}{\partial t} g_n(t, x) \right) \omega(t, x) dx + \int f(x) g_n(t, x) \left(\frac{\partial}{\partial t} \omega(t, x) \right) dx \\ &= -t^{-1} (2n+1)^{\frac{1}{2}} \left[n(2n+1)^{-\frac{1}{2}} c_n(t) + (2n-1)^{\frac{1}{2}} c_{n-1}(t) + (2n-3)^{\frac{1}{2}} c_{n-2}(t) + \dots \right] \\ &\quad - t^{-1} c_n(t) + t^{-1} f(t) g_n(t, t) \\ &= -t^{-1} (2n+1)^{\frac{1}{2}} \left[(n+1)(2n+1)^{-\frac{1}{2}} c_n(t) + (2n-1)^{\frac{1}{2}} c_{n-1}(t) + (2n-3)^{\frac{1}{2}} c_{n-2}(t) + \dots \right] \\ &\quad + t^{-1} (2n+1)^{\frac{1}{2}} f(t)\end{aligned}$$

where we have used $g_n(t, t) = (2n + 1)^{\frac{1}{2}} P_n(1) = (2n + 1)^{\frac{1}{2}}$. Vectorizing this yields equation (4.2):

$$\begin{aligned} c'(t) &= -\frac{1}{t} \mathbf{A}c(t) + \frac{1}{t} \mathbf{B}f(t) \\ \mathbf{A}_{nk} &= \begin{cases} (2n + 1)^{1/2}(2k + 1)^{1/2} & \text{if } n > k \\ n + 1 & \text{if } n = k, \\ 0 & \text{if } n < k \end{cases} \\ \mathbf{B}_n &= (2n + 1)^{\frac{1}{2}} \end{aligned} \quad (\text{D.12})$$

Alternatively, we can write this as

$$\frac{d}{dt}c(t) = -t^{-1} \mathbf{D} [\mathbf{M}\mathbf{D}^{-1}c(t) + \mathbf{1}f(t)], \quad (\text{D.13})$$

where $\mathbf{D} := \text{diag} \left[(2n + 1)^{\frac{1}{2}} \right]_{n=0}^{N-1}$, $\mathbf{1}$ is the all ones vector, and the state matrix \mathbf{M} is

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 1 & 2 & 0 & 0 & \dots & 0 \\ 1 & 3 & 3 & 0 & \dots & 0 \\ 1 & 3 & 5 & 4 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 3 & 5 & 7 & \dots & N \end{bmatrix}, \quad \text{that is,} \quad \mathbf{M}_{nk} = \begin{cases} 2k + 1 & \text{if } k < n \\ k + 1 & \text{if } k = n \\ 0 & \text{if } k > n \end{cases}$$

Equation (D.12) is a linear dynamical system, except dilated by a time-varying factor t^{-1} , which arises from the scaled measure.

Reconstruction

By equation (D.6), at every time t we have

$$\begin{aligned} f(x) \approx g^{(t)}(x) &= \sum_n c_n(t) g_n(t, x). \\ &= \sum_n c_n(t) (2n + 1)^{\frac{1}{2}} P_n \left(\frac{2x}{t} - 1 \right). \end{aligned}$$

D.3 HIPPO-LegS Theoretical Properties

D.3.1 Timescale equivariance

Proof of Lemma 4.5. Let $\tilde{f}(t) = f(\alpha t)$. Let $c = \text{proj } f$ and $\tilde{c} = \text{proj } \tilde{f}$. By the HIPPO equation (D.5) update and the basis instantiation for LegS (equation (D.11)),

$$\begin{aligned}\tilde{c}_n(t) &= \langle \tilde{f}, g_n^{(t)} \rangle_{\mu^{(t)}} \\ &= \int \tilde{f}(t)(2n+1)^{\frac{1}{2}} P_n\left(2\frac{x}{t}-1\right) \frac{1}{t} \mathbb{I}_{[0,1]}\left(\frac{x}{t}\right) dx \\ &= \int f(\alpha t)(2n+1)^{\frac{1}{2}} P_n\left(2\frac{x}{t}-1\right) \frac{1}{t} \mathbb{I}_{[0,1]}\left(\frac{x}{t}\right) dx \\ &= \int f(\alpha t)(2n+1)^{\frac{1}{2}} P_n\left(2\frac{x}{\alpha t}-1\right) \frac{1}{\alpha t} \mathbb{I}_{[0,1]}\left(\frac{x}{\alpha t}\right) dx \\ &= c_n(\alpha t).\end{aligned}$$

The second-to-last equality uses the change of variables $x \mapsto \frac{x}{\alpha}$. □

D.3.2 Speed

In this section we work out the fast update rules according to generalized bilinear transform (GBT) discretization, which generalizes the forward Euler, backward Euler, and bilinear discretizations (cf. Appendix A.1). For the GBT discretization, we must be able to perform matrix-vector multiplication by $\mathbf{I} + \delta \mathbf{A}$ and $(\mathbf{I} - \delta \mathbf{A})^{-1}$ where δ is any multiple of the step size Δ .

It is easily seen that the LegS update rule involves a matrix \mathbf{A} of the following form (Theorem 4.3): $\mathbf{A} = \mathbf{D}_1(\mathbf{L} + \mathbf{D}_0)\mathbf{D}_2$, where \mathbf{L} is the all 1 lower triangular matrix and $\mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2$ are diagonal. Clearly, $\mathbf{I} + \delta \mathbf{A}$ is efficient (only requiring $O(N)$ operations), as it only involves matrix-vector multiplication by diagonals $\mathbf{D}_0, \mathbf{D}_1, \mathbf{D}_2$, or multiplication by \mathbf{L} which is the **cumsum** operation.

Now we consider multiplication by the inverse $(\mathbf{I} + \delta \mathbf{A})^{-1}$ (the minus sign can be absorbed into δ). Write

$$\begin{aligned}(\mathbf{I} + \delta \mathbf{D}_1(\mathbf{L} + \mathbf{D}_0)\mathbf{D}_2)^{-1} &= (\mathbf{D}_1(\mathbf{D}_1^{-1}\mathbf{D}_2^{-1} + \delta(\mathbf{L} + \mathbf{D}_0))\mathbf{D}_2)^{-1} \\ &= \delta^{-1}\mathbf{D}_2^{-1}(\delta^{-1}\mathbf{D}_1^{-1}\mathbf{D}_2^{-1} + \mathbf{D}_0 + \mathbf{L})^{-1}\mathbf{D}_1^{-1}\end{aligned}$$

Since diagonal multiplication is efficient, the crucial operation is inversion multiplication by

a matrix of the form $\mathbf{L} + \mathbf{D}$.

Consider solving the equation $(\mathbf{L} + \mathbf{D})x = y$. This implies $x_0 + \dots + x_{k-1} = y_k - (1 + d_k)x_k$. The solution is

$$\begin{aligned} x_0 &= \frac{y_0}{1 + d_0} \\ x_k &= \frac{y_k - x_0 - \dots - x_{k-1}}{1 + d_k} \end{aligned}$$

Define $s_k = x_0 + \dots + x_k$. Then

$$s_k = s_{k-1} + x_k = s_{k-1} + \frac{y_k - s_{k-1}}{1 + d_k} = \frac{y_k + d_k s_{k-1}}{1 + d_k} = \frac{d_k}{1 + d_k} s_{k-1} + \frac{y_k}{1 + d_k}.$$

Finally, consider how to calculate a recurrence of the following form efficiently.

$$x_0 = \beta_0, x_k = \alpha_k x_{k-1} + \beta_k.$$

This update rule can also be written

$$\frac{x_k}{\alpha_k \dots \alpha_1} = \frac{x_{k-1}}{\alpha_{k-1} \dots \alpha_1} + \frac{\beta_k}{\alpha_k \dots \alpha_1}.$$

Evidently x can be computed in a vectorized way as

$$x = \text{cumsum}(\beta / \text{cumprod}(\alpha)) \cdot \text{cumprod}(\alpha).$$

This is an $O(N)$ computation.

D.3.3 Gradient Norms

We analyze the discrete time case under the Euler discretization (Appendix A.1), where the HIPPO-LegS recurrent update is equation (4.3), restated here for convenience:

$$c_{k+1} = \left(1 - \frac{\mathbf{A}}{k}\right) c_k + \frac{1}{k} \mathbf{B} f_k.$$

These gradient asymptotics hold under other discretizations.

We will show that

Proposition D.1. *For any times $k < \ell$, the gradient norm of the HIPPO-LegS operator for the output at time $\ell + 1$ with respect to input at time k is $\left\| \frac{\partial c_{\ell+1}}{\partial f_k} \right\| = \Theta(1/\ell)$.*

Proof. We take N to be a constant.

Without loss of generality assume $k > 2$, as the gradient change for a single initial step is bounded. By unrolling the recurrence (4.3), the dependence of $c_{\ell+1}$ on c_k and f_k, \dots, f_ℓ can be made explicit:

$$\begin{aligned} c_{\ell+1} = & \left(\mathbf{I} - \frac{\mathbf{A}}{\ell} \right) \dots \left(\mathbf{I} - \frac{\mathbf{A}}{k} \right) c_k \\ & + \left(\mathbf{I} - \frac{\mathbf{A}}{\ell} \right) \dots \left(\mathbf{I} - \frac{\mathbf{A}}{k+1} \right) \frac{\mathbf{B}}{k} f_k \\ & + \left(\mathbf{I} - \frac{\mathbf{A}}{\ell} \right) \dots \left(\mathbf{I} - \frac{\mathbf{A}}{k+2} \right) \frac{\mathbf{B}}{k+1} f_{k+1} \\ & \vdots \\ & + \left(\mathbf{I} - \frac{\mathbf{A}}{\ell} \right) \frac{\mathbf{B}}{\ell-1} f_{\ell-1} \\ & + \frac{\mathbf{B}}{\ell} f_\ell. \end{aligned}$$

Therefore

$$\frac{\partial c_{\ell+1}}{\partial f_k} = \left(\mathbf{I} - \frac{\mathbf{A}}{\ell} \right) \dots \left(\mathbf{I} - \frac{\mathbf{A}}{k+1} \right) \frac{\mathbf{B}}{k}.$$

Notice that \mathbf{A} has distinct eigenvalues $1, 2, \dots, N$, since those are the elements of its diagonal and \mathbf{A} is triangular (Theorem 4.3). Thus the matrices $\mathbf{I} - \frac{\mathbf{A}}{\ell}, \dots, \mathbf{I} - \frac{\mathbf{A}}{k+1}$ are diagonalizable with a common change of basis. The gradient then has the form $\mathbf{P}\mathbf{D}\mathbf{P}^{-1}\mathbf{B}$ for some invertible matrix \mathbf{P} and some diagonal matrix \mathbf{D} . Its norm is therefore bounded from below (up to constant) by the smallest singular value of \mathbf{P} and $\|\mathbf{P}^{-1}\mathbf{B}\|$, both of which are nonzero constants, and the largest diagonal entry of \mathbf{D} . It thus suffices to bound this largest diagonal entry of \mathbf{D} , which is the largest eigenvalue of this product,

$$\rho = \left(1 - \frac{1}{\ell} \right) \dots \left(1 - \frac{1}{k+1} \right) \frac{1}{k}.$$

The problem reduces to showing that $\rho = \Theta(1/\ell)$.

We will use the following facts about the function $\log(1 - \frac{1}{x})$. First, it is an increasing

function, so

$$\log\left(1 - \frac{1}{x}\right) \geq \int_{x-1}^x \log\left(1 - \frac{1}{\lambda}\right) d\lambda.$$

Second, its antiderivative is

$$\begin{aligned} \int \log\left(1 - \frac{1}{x}\right) dx &= \int \log(x-1) - \log(x) = (x-1)\log(x-1) - x\log(x) \\ &= x\log\left(1 - \frac{1}{x}\right) - \log(x-1). \end{aligned}$$

Therefore, we have

$$\begin{aligned} \log\left(1 - \frac{1}{\ell}\right) \dots \left(1 - \frac{1}{k+1}\right) &= \sum_{i=k+1}^{\ell} \log\left(1 - \frac{1}{i}\right) \\ &\geq \sum_{i=k+1}^{\ell} \int_{i-1}^i \log\left(1 - \frac{1}{x}\right) dx \\ &= \int_k^{\ell} \log\left(1 - \frac{1}{x}\right) dx \\ &= [(x-1)\log(x-1) - x\log(x)]|_k^{\ell} \\ &= \ell\log\left(1 - \frac{1}{\ell}\right) - \log(\ell-1) \\ &\quad - \left(k\log\left(1 - \frac{1}{k}\right) - \log(k-1)\right). \end{aligned}$$

Finally, note that $x\log\left(1 - \frac{1}{x}\right)$ is an increasing function, and bounded from above since it is negative, so it is $\Theta(1)$ (this can also be seen from its Taylor expansion). Thus we have

$$\log\rho \geq \Theta(1) - \log(\ell-1) + \log(k-1) - \log(k),$$

Furthermore, all inequalities are asymptotically tight, so that $\rho = \Theta(1/\ell)$ as desired. \square

D.3.4 Function Approximation Error

Proof of Proposition 4.8. Fix a time t . HIPPO-LegS uses the measure $\omega(t, x) = \frac{1}{t}\mathbb{I}_{[0,t]}$ and the polynomial basis $p_n(t, x) = (2n+1)^{\frac{1}{2}}P_n\left(\frac{2x}{t}-1\right)$. Let $c_n(t) = \langle f_{\leq t}, p_n^{(t)} \rangle_{\mu^{(t)}}$ for

$n = 0, 1, \dots$. Then the projection $g^{(t)}$ is obtained by linear combinations of the basis functions, with $c_n(t)$ as coefficients:

$$g^{(t)} = \sum_{n=0}^{N-1} c_n(t) p_n^{(t)}.$$

Since $p_n^{(t)}$ forms an orthonormal basis of the Hilbert space defined by the inner product $\langle \cdot, \cdot \rangle_{\mu^{(t)}}$ [26], by Parseval's identity,

$$\left\| f_{\leq t} - g^{(t)} \right\|_{\mu^{(t)}}^2 = \sum_{n=N}^{\infty} c_n^2(t).$$

To bound the error $\|f_{\leq t} - g^{(t)}\|_{\mu^{(t)}}$, it suffices to bound the sum of the squares of the high-order coefficients $c_n(t)$ for $n = N, N+1, \dots$. We will bound each coefficient by integration by parts.

We first simplify the expression for $c_n(t)$. For any $n \geq 1$, we have

$$\begin{aligned} c_n(t) &= \langle f_{\leq t}, p_n^{(t)} \rangle_{\mu^{(t)}} \\ &= \frac{1}{t} (2n+1)^{\frac{1}{2}} \int_0^t f(x) P_n \left(\frac{2x}{t} - 1 \right) dx \\ &= \frac{(2n+1)^{\frac{1}{2}}}{2} \int_{-1}^1 f \left(\frac{1+x}{2} t \right) P_n(x) dx \quad (\text{change of variable } x \rightarrow \frac{1+x}{2} t). \end{aligned}$$

As $P_n(x) = \frac{1}{2n+1} \frac{d}{dx} (P_{n+1}(x) - P_{n-1}(x))$ (cf. Appendix A.2.1), integration by parts yields:

$$\begin{aligned} c_n(t) &= \frac{(2n+1)^{\frac{1}{2}}}{2} \left[f \left(\frac{1+x}{2} t \right) \frac{1}{2n+1} (P_{n+1}(x) - P_{n-1}(x)) \right] \Big|_{-1}^1 \\ &\quad - \frac{(2n+1)^{\frac{1}{2}}}{2} \int_{-1}^1 \frac{t}{2} f' \left(\frac{1+x}{2} t \right) \frac{1}{2n+1} (P_{n+1}(x) - P_{n-1}(x)) dx. \end{aligned}$$

Notice that the boundary term is zero, since $P_{n+1}(1) = P_{n-1}(1) = 1$ and $P_{n+1}(-1) = P_{n-1}(-1) = \pm 1$ (either both 1 or both -1 depending on whether n is odd or even). Hence:

$$c_n(t) = -\frac{1}{4} \cdot \frac{1}{(2n+1)^{\frac{1}{2}}} \cdot t \int_{-1}^1 f' \left(\frac{1+x}{2} t \right) (P_{n+1}(x) - P_{n-1}(x)) dx.$$

Now suppose that f is L -Lipschitz, which implies that $|f'| \leq L$. Then

$$\begin{aligned} c_n^2(t) &\leq t^2 L^2 \frac{1}{16} \cdot \frac{1}{2n+1} \left[\int_{-1}^1 |P_{n+1}(x) - P_{n-1}(x)| dx \right]^2 \\ &\leq t^2 L^2 \frac{1}{16} \cdot \frac{1}{2n+1} \cdot 2 \int_{-1}^1 (P_{n+1}(x) - P_{n-1}(x))^2 dx \quad (\text{Cauchy-Schwarz}) \\ &= t^2 L^2 \frac{1}{8} \frac{1}{2n+1} \left[\int_{-1}^1 P_{n+1}^2(x) dx + \int_{-1}^1 P_{n-1}^2(x) dx \right] \quad (P_{n+1} \text{ and } P_{n-1} \text{ are orthogonal}) \\ &= t^2 L^2 \frac{1}{8} \frac{1}{2n+1} \left[\frac{2}{2n+3} + \frac{2}{2n-1} \right] \\ &= O(1)t^2 L^2 \frac{1}{n^2}. \end{aligned}$$

Summing for all $n \geq N$ yields:

$$\left\| f_{\leq t} - g^{(t)} \right\|_{\mu^{(t)}}^2 = \sum_{n=N}^{\infty} c_n^2(t) = O(1)t^2 L^2 \sum_{n=N}^{\infty} \frac{1}{n^2} = O(1)t^2 L^2 \frac{1}{N}.$$

We then obtain that $\|f_{\leq t} - g^{(t)}\|_{\mu^{(t)}} = O(tL/\sqrt{N})$ as claimed.

Now suppose that f has k derivatives and the k -th derivative is bounded. The argument is similar to the one above where we integrate by parts k times. We sketch this argument here.

Take k to be a constant, and let $n \geq k$. Applying integration by parts k times, noting that all the boundary terms are zero, gives:

$$c_n(t) = O(1)(2n+1)^{\frac{1}{2}} t^k \int_{-1}^1 f^{(k)} \left(\frac{1+x}{2} t \right) q_k(x) dx,$$

where $q_k(x)$ is a polynomial such that $\frac{d^k}{dx^k} q_k(x) = P_n(x)$. Then since $f^{(k)}$ is bounded, $|c_n(t)| = O(1)(2n+1)^{\frac{1}{2}} \int_{-1}^1 |q_k(x)| dx$, and so

$$c_n^2(t) = O(1)t^{2k}(2n+1) \left[\int_{-1}^1 |q_k(x)| dx \right]^2 = O(1)t^{2k}(2n+1) \int_{-1}^1 q_k^2(x) dx \quad (\text{Cauchy-Schwarz}).$$

It remains to bound $\int_{-1}^1 q_k^2(x) dx$. Using the fact that $\frac{d}{dx} P_n(x) = \frac{1}{2n+1} (P_{n+1}(x) - P_{n-1}(x))$

repeatedly, we have:

$$\begin{aligned}
q_1 &= \frac{1}{2n+1}(P_{n+1} - P_{n-1}) = \frac{1}{n+O(1)} \cdot \frac{1}{2}(P_{n+1} - P_{n-1}) \\
q_2 &= \frac{1}{(n+O(1))^2} \frac{1}{2^2}(P_{n+2} - P_n - P_n + P_{n-2}) = \frac{1}{(n+O(1))^2} \frac{1}{2^2}(P_{n+2} - 2P_n + P_{n-2}) \\
q_3 &= \frac{1}{(n+O(1))^3} \frac{1}{2^3}(P_{n+3} - P_{n+1} - 2P_{n+1} + 2P_{n-1} + P_{n-1} - P_{n-3}) \\
&= \frac{1}{(n+O(1))^3} \frac{1}{2^3}(P_{n+3} - 3P_{n+1} + 3P_{n-1} - P_{n-3}) \\
&\vdots
\end{aligned}$$

In general, when we expand out $\int_{-1}^1 q_k^2(x) dx$, since the P_m 's are orthogonal, we get $k+1$ terms of the form $\frac{1}{(n+O(1))^{2k}} \frac{1}{2^{2k}} \binom{k}{l}^2 \int_{-1}^1 P_m^2(x) dx$ for k different values of m in the range $[n-k, n+k]$, and l goes from 0 to k . For each m , $\int_{-1}^1 P_m^2(x) dx = \frac{1}{n+O(1)}$, and $\sum_{l=0}^k \binom{k}{l}^2 = \binom{2k}{k}$. Summing up all $k+1$ terms yields

$$\int_{-1}^1 q_k^2(x) dx = \frac{1}{(n+O(1))^{2k+1}} \frac{1}{2^k} \binom{2k}{k}.$$

By Stirling's approximation, $\binom{2k}{k} = O(1)4^k$, so $\int_{-1}^1 q_k^2(x) dx = \frac{O(1)2^k}{(n+O(1))^{2k+1}}$. Noting that k is a constant, plugging this into the bound for $c_n^2(t)$:

$$c_n^2(t) = O(1)t^{2k}(2n+1) \frac{O(1)2^k}{(n+O(1))^{2k+1}} = O(1)t^{2k} \frac{1}{n^{2k}}.$$

Summing for all $n \geq N$ yields:

$$\left\| f_{\leq t} - g^{(t)} \right\|_{\mu^{(t)}}^2 = \sum_{n=N}^{\infty} c_n^2(t) = O(1)t^{2k} \sum_{n=N}^{\infty} \frac{1}{n^{2k}} = O(1)t^{2k} \frac{1}{N^{2k-1}}.$$

We then obtain that $\left\| f_{\leq t} - g^{(t)} \right\|_{\mu^{(t)}} = O(t^k N^{-k+1/2})$ as claimed. \square

Remark D.2. *The approximation error of Legendre polynomials reduces to how fast the Legendre coefficients decay, subjected to the smoothness assumption of the input function. This result is analogous to the classical result in Fourier analysis, where the n -th Fourier coefficients decay as $O(n^{-k})$ if the input function has order- k bounded derivatives [113]. That result is also proved by integration by parts.*

D.4 Experiment Details and Additional Results

Appendix D.4.1 and Appendix D.4.2 contain details for Section 4.4.1 and Section 4.4.3 respectively.

D.4.1 Online Function Approximation and Speed Benchmark

Task Details

The task is to reconstruct an input function (as a discrete sequence) based on some hidden state produced after the model has traversed the input function. This is the same problem setup as in Section 4.2.1; mathematical details for the online approximation and reconstruction are in Appendix D.1. The input function is randomly sampled from a continuous-time band-limited white noise process, with length 10^6 . The sampling step size is $\Delta = 10^{-4}$, and the signal band limit is 1Hz.

Models

We compare HIPPO-LegS, LMU, and LSTM. The HIPPO-LegS and LMU model only consists of the memory update and not the additional RNN architecture. The function is reconstructed from the coefficients using the formula in Appendix D.2, so no training is required. For LSTM, we use a linear decoder to reconstruct the function from the LSTM hidden states and cell states, trained on a collection of 100 sequences. All models use $N = 256$ hidden units. The LSTM uses the L_2 loss. The HIPPO methods including LMU follow the fixed dynamics of Theorem 4.2 and Theorem 4.3.

Speed Benchmark

We measure the inference time of HIPPO-LegS, LMU, and LSTM, in single-threaded mode on a server Intel Xeon CPU E5-2690 v4 at 2.60GHz.

D.4.2 Trajectory Classification

Dataset Details

The Character Trajectories dataset [9] from the UCI machine learning repository [52] consists of pen tip trajectories recorded from writing individual characters. The trajectories were captured at 200Hz and data was normalized and smoothed. Input is 3-dimensional (x and y positions, and pen tip force), and there are 20 possible outputs (number of classes). Models are trained using the cross-entropy loss. The dataset contains 2858 time series.

The length of the sequences is variable, ranging up to 182. We use a train-val-test split of 70%-15%-15%.

Methods

RNN baselines include the LSTM [86], GRU [31], and LMU [218]. Our implementations of these used 256 hidden units each.

The GRU-D [23] is a method for handling missing values in time series that computes a decay between observations. The ODE-RNN [176] and Neural CDE (NCDE) [107] baselines are state-of-the-art neural ODE methods, also designed to handle irregularly-sampled time series. Our GRU-D, ODE-RNN, and Neural CDE baselines used code from Kidger et al. [107], inheriting the hyperparameters for those methods.

All methods trained for 100 epochs.

Timescale Mis-specification

The goal of this experiment is to investigate the performance of models when the timescale is mis-specified between train and evaluation time, leading to distribution shift. We considered the following two standard types of time series:

1. Sequences sampled at a fixed rate
2. Irregularly-sampled time series (i.e., missing values) with timestamps

Timescale shift is emulated in the corresponding ways, which can be interpreted as different sampling rates or trajectory speeds.

1. Either the train or evaluation sequences are downsampled by a factor of 2
2. The train or evaluation timestamps are halved.¹

The first scenario in each corresponds to the original sequence being sampled at 100Hz instead of 200Hz; alternatively, it is equivalent to the writer drawing twice as fast. Thus, these scenarios correspond to a train → evaluation timescale shift of 100Hz → 200Hz and 200Hz → 100Hz respectively.

Note that models are unable to obviously tell that there is timescale shift. For example, in the first scenario, shorter or longer sequences can be attributed to the variability of

¹Instead of the train timestamps being halved, equivalently the evaluation timestamps can be doubled.

sequence lengths in the original dataset. In the second scenario, the timestamps have different distributions, but this can correspond to different rates of missing data, which the baselines for irregularly-sampled data are able to address.

Appendix E

Appendix for Chapter 5

We furnish the missing proofs from Section 5.1 in Appendix E.1. We will describe our general framework and results in Appendix E.2, and prove the results in Sections 5.2 to 5.4 in Appendices E.3 to E.5 respectively.

E.1 Proofs from Background

This corresponds to results from Section 5.1.

Proof of Proposition 5.2. Suppose for the sake of contradiction that there is a second basis and measure q_n, μ such that q_n is complete and orthogonal w.r.t. μ , and $K_n = q_n\mu$. By completeness, there are coefficients $c_{\ell,k}$ such that

$$p_\ell = \sum_k c_{\ell,k} q_k.$$

Then

$$\int p_\ell q_j \mu = \int \sum_k c_{\ell,k} q_k q_j \mu = \sum_k c_{\ell,k} \delta_{kj} = c_{\ell,j}.$$

But $q_j \mu = K_j = p_j \omega$, so

$$\int p_\ell q_j \mu = \int p_\ell p_j \omega = \delta_{\ell j}.$$

So $c_{\ell,j} = \delta_{\ell,j}$ which implies that $p_\ell = q_\ell$ for all ℓ , as desired. \square

Proof of Proposition 5.4. The SSM kernels are $K_n(t) = e^{-t(n+1)} \mathbf{B}_n$. Assume $\mathbf{B}_n \neq 0$ so that the kernels are not degenerate.

Suppose for the sake of contradiction that this was a TO-SSM with measure $\omega(t)$. Then we must have

$$\int K_n(s) K_m(s) \omega(t)^{-1} ds = \delta_{n,m}$$

Plugging in $n = 1, m = 1$ and $n = 0, m = 2$ gives

$$\begin{aligned} 1 &= \int e^{-2t} \mathbf{B}_1 e^{-2t} \mathbf{B}_1 \omega(t)^{-1} ds = \mathbf{B}_1 \mathbf{B}_1 \int e^{-4t} \omega(t)^{-1} ds \\ 0 &= \int e^{-1t} \mathbf{B}_0 e^{-3t} \mathbf{B}_2 \omega(t)^{-1} ds = \mathbf{B}_0 \mathbf{B}_2 \int e^{-4t} \omega(t)^{-1} ds \end{aligned}$$

This is clearly a contradiction. \square

E.2 General Theory

Consider a measure supported on $[0, 1]$ with density $\omega(t)\mathbb{I}(t)$, where $\mathbb{I}(t)$ is the indicator function for membership in the interval $[0, 1]$. Let the measure be equipped with a set of orthonormal basis functions p_0, \dots, p_{N-1} , i.e.

$$\int p_j(s) p_k(s) \omega(s) \mathbb{I}(s) ds = \delta_{jk}, \tag{E.1}$$

where the integrals in this paper are over the range $[-\infty, \infty]$, unless stated otherwise.

This is sufficient to derive an O-SSM based on the HIPPO technique. The generalized HIPPO framework demonstrates how to build (T)O-SSMs utilizing *time warping* to shape the time interval and *tilting* to construct new sets of orthogonal basis functions.

Given an general interval $[\ell, r]$, we will use the notation $\mathbb{I}[\ell, r]$ to denote the indicator function for the interval $[\ell, r]$ —we will drop the interval if $\ell = 0, r = 1$.

We will need the notion of a “*time warping*” function $\bar{\sigma}$ as follows:

Definition E.1. A time warping function is defined as

$$\bar{\sigma}(t, s) : (-\infty, t] \rightarrow [0, 1]$$

such that $\bar{\sigma}(t, t) = 1$.

We will be using a special case of time-warping function, which we say has a discontinuity at t_0 for some $t_0 \in (-\infty, t]$:

$$\bar{\sigma}(t, s) = \mathbb{I}[t_0, t]\sigma(t, s), \quad (\text{E.2})$$

such that

$$\frac{\partial}{\partial t} \left(\frac{\partial}{\partial s} \sigma_s(t, s) \right) = c(t) \frac{\partial}{\partial s} \sigma(t, s). \quad (\text{E.3})$$

We allow for $t_0 = -\infty$, in which case we think of the interval $[t_0, t]$ as $(-\infty, t]$.

Before proceeding, let us clarify our notation. We will use σ_t and σ_s to denote the partial derivatives $\frac{\partial}{\partial t} \sigma(t, s)$ and $\frac{\partial}{\partial s} \sigma(t, s)$ respectively. We will drop the parameters (t, s) and use f instead of $f(t, s)$ when it is clear from context to reduce notational clutter. Further, we will extend this notation to function composition, i.e. write $g \circ f(t, s)$ as $g(f)$ and function product, i.e. use fgh instead of $f(t, s)g(t, s)h(t, s)$. Finally, we'll shorten $fgh \circ \phi(t, s)$ as $fgh(\phi)$.

We also define the tilting χ and show that regardless of warping, we can construct a new orthogonal basis (note that the result holds for warping functions as in (E.2) and not just those as in (E.3)).

Lemma E.2. For the set of orthonormal functions $\{p_n\}_{n=0}^{N-1}$ orthogonal over measure ωI , the set of basis functions

$$q_k^t(\sigma(t, s)) = \chi(t, s)p_k(\sigma(t, s))$$

are orthogonal over the measure

$$\mu(t, s) = \omega(\sigma(t, s))\mathbb{I}[t_0, t](s)\sigma_s(t, s)\chi(t, s)^{-2}$$

for time-warping function σ satisfying (E.2) and any $\chi(t, s)$ that is non-zero in its support.

Proof. Consider the following sequence of equalities:

$$\begin{aligned}
\int p_j(\sigma) p_k(\sigma) \omega(\sigma) \mathbb{I}[t_0, t] \sigma_s ds &= \int_{t_0}^t p_j(\sigma) p_k(\sigma) \omega(\sigma) \sigma_s ds \\
&= \int_{\sigma(t_0)}^{\sigma(t,t)} p_j(y) p_k(y) \omega(y) dy \\
&= \int_{\sigma(t_0)}^{\sigma(t,t)} p_j(y) p_k(y) \omega(y) dy \\
&= \int_0^1 p_j(y) p_k(y) \omega(y) dy \\
&= \int p_j(y) p_k(y) \omega(y) \mathbb{I}(y) dy \\
&= \delta_{jk}.
\end{aligned}$$

In the above, the second equality follows from the substitution $y \leftarrow \sigma(t, s)$ and hence $dy = \sigma_s ds$ and the final equality follows from (E.1). Then since $\chi(t, s)$ is always non-zero, we have

$$\int (\chi p_j(\sigma)) (\chi p_k(\sigma)) \omega(\sigma) \mathbb{I}[t_0, t] \sigma_s \chi^{-2} ds = \delta_{jk},$$

as desired. \square

Without loss of generality, we can split χ into a product

$$\chi(t, s) = \frac{1}{\psi(\sigma(t, s)) \phi(t, s)} \quad (\text{E.4})$$

of one part that depends on σ and another arbitrary component.

Time Warped HIPPO. Since we have an orthonormal basis and measure, we can try to derive the (T)O-SSM. For a given input signal $u(t)$, the HIPPO coefficients are defined as the projections.

$$\begin{aligned}
x_n(t) &= \langle u, \chi p_n \rangle_\mu \\
&= \int u(s) \cdot \chi \cdot (p_n \omega)(\sigma) \mathbb{I}[t_0, t] \sigma_s \chi^{-2} ds
\end{aligned}$$

defined as inner product of $u(t)$ with the tilted basis functions χp_n with respect to the measure μ as defined in Lemma E.2. For additional convenience, we use the decomposition $\chi = \psi^{-1}\phi^{-1}$ from (E.4) to get:

$$x_n(t) = \int u(s) \cdot (p_n \omega \psi)(\sigma) \mathbb{I}[t_0, t] \sigma_s \phi ds. \quad (\text{E.5})$$

The HIPPO technique is to differentiate through this integral in a way such that it can be related back to $x_n(t)$ and other $x_k(t)$. We require for every n , we require that there are a set of coefficients $\{\gamma_{nk}\}_{k=0}^{N-1}$ such that

$$\sigma_t(p_n \omega \psi)'(\sigma) = \sum_{k=0}^{N-1} \gamma_{nk} (p_n \omega \psi)(\sigma) \quad (\text{E.6})$$

and for tilting component ϕ

$$\frac{d}{dt} \phi(t, s) = d(t) \phi(t, s). \quad (\text{E.7})$$

Theorem E.3. Consider a set of basis functions p_n orthogonal over ω , time warping $\bar{\sigma}(t, s)$ as in (E.2), (E.3), and tilting χ as in (E.4) and (E.7) with the functions $\sigma, p_n, \omega, \psi$ obeying (E.6). If $\frac{dt_0}{dt} \neq 0$, further assume that for some vector \mathbf{A}' , we have as $N \rightarrow \infty$,

$$u(t_0) = \bar{c} \sum_{k=0}^{N-1} \mathbf{A}'_k \cdot x_k(t) + \bar{d}u(t). \quad (\text{E.8})$$

Then $(\mathbf{A}^0 + (c(t) + d(t))\mathbf{I} - \bar{c}\mathbf{D}(\mathbf{A}')^\top, \mathbf{B} - \bar{d}\mathbf{D})$ is an O-SSM for basis functions $\chi p_n(\sigma)$ with measure $\omega \mathbb{I}[t_0, t] \sigma_s \chi^{-2}$ where

$$\mathbf{A}'_{nk} = \gamma_{nk}$$

with γ_{nk} as in (E.6),

$$\mathbf{D}_n = (p_n \omega \psi)(\sigma(t, t_0))(\sigma_s \phi)(t, t_0) \cdot \frac{dt_0}{dt},$$

and

$$\mathbf{B}_n = (p_n \omega \psi)(1)(\sigma_s \phi)(t, t).$$

Proof. Applying the Leibniz rule to (E.5), we get

$$x'_n(t) = x_n^{(0)}(t) + x_n^{(1)}(t) + x_n^{(2)}(t) + x_n^{(3)}(t),$$

where

$$\begin{aligned} x_n^{(0)}(t) &= \int u(s) \cdot \sigma_t(p_n \omega \psi)'(\sigma) \mathbb{I}[t_0, t] \sigma_s \phi \, ds \\ x_n^{(1)}(t) &= \int u(s) \cdot (p_n \omega \psi)(\sigma) \mathbb{I}[t_0, t] \left[\frac{\partial}{\partial t} (\sigma_s \phi) \right] \, ds \end{aligned}$$

and the $x_n^{(2)}(t) + x_n^{(3)}(t)$ terms capture the term we get when differentiating $\mathbb{I}[t_0, t]$.

Let us consider each term separately. The first term

$$x_n^{(0)}(t) = \int u(s) \cdot \sigma_t(p_n \omega \psi)'(\sigma) \mathbb{I}[t_0, t] \sigma_s \phi \, ds \quad (\text{E.9})$$

corresponds to the differentiation of the basis functions and measure. In order to relate this to $\{x_k(t)\}$, it suffices that $\sigma_t(p_n \omega \psi)'(\sigma)$ satisfies (E.6) which implies that when we vectorize this, we get $x^{(0)}(t) = \mathbf{A}^0 \cdot x(t)$.

For additional warping and tilting terms, we consider

$$x_n^{(1)}(t) = \int u(s) \cdot (p_n \omega \psi)(\sigma) \mathbb{I}[t_0, t] \left[\frac{\partial}{\partial t} (\sigma_s \phi) \right] \, ds.$$

To reduce this term to $x_n(t)$, recall from (E.3) that

$$\partial_t(\sigma_s) = c(t) \sigma_s.$$

Then the above and (E.7) imply

$$\partial_t(\sigma_s \phi) = c(t)(\sigma_s \phi) + d(t)(\sigma_s \phi)$$

where $c(t), d(t)$ are defined as in (E.3) and (E.7).

We will end up with $x_n^{(1)}(t) = (c(t) + d(t))x_n(t)$. This leads to the vectorized form $x^{(1)}(t) = (c(t) + d(t))\mathbf{I}x(t)$.

We now need to handle

$$x_n^{(2)}(t) + x_n^{(3)}(t) = \int u(s) \cdot (p_n \omega \psi)(\sigma) \left[\frac{\partial}{\partial t} \mathbb{I}[t_0, t] \right] (\sigma_s \phi) ds. \quad (\text{E.10})$$

For the above note that

$$\mathbb{I}[t_0, t](s) = H(s - t_0) - H(s - t),$$

where $H(x)$ is the ‘‘heaviside step function.’’ It is known that $H'(x) = \delta(x)$, which implies

$$\frac{\partial}{\partial t} \mathbb{I}[t_0, t] = \delta(s - t) - \frac{dt_0}{dt} \delta(s - t_0).$$

Using the above in RHS of (E.10), we separate out $x_n^{(2)}(t)$ and $x_n^{(3)}(t)$ as follows. First, define

$$\begin{aligned} x_n^{(2)}(t) &= \int u(s) \cdot (p_n \omega \psi)(\sigma) \delta(s - t) \sigma_s \phi ds \\ &= u(t) \cdot (p_n \omega \psi)(\sigma(t, t))(\sigma_s \phi)(t, t) \\ &= u(t) \cdot (p_n \omega \psi)(1)(\sigma_s \phi)(t, t). \end{aligned}$$

In the last equality, we have used the fact that $\sigma(t, t) = \bar{\sigma}(t, 1) = 1$ by definition. It follows that in vectorized form we have $x^{(2)}(t) = \mathbf{B}u(t)$.

Finally, define

$$\begin{aligned} x_n^{(3)}(t) &= - \int u(s) \cdot (p_n \omega \psi)(\sigma) \delta(s - t_0) \frac{dt_0}{dt} \sigma_s \phi ds \\ &= -u(t_0) \cdot (p_n \omega \psi)(\sigma(t, t_0)) (\sigma_s \phi)(t, t_0) \cdot \frac{dt_0}{dt} \end{aligned}$$

If $\frac{dt_0}{dt} = 0$, then we have $\mathbf{D} = \mathbf{0}$ and hence we have $x^{(3)}(t) = \mathbf{0} = -\bar{c}\mathbf{D}(\mathbf{A}')^\top x(t) - \bar{d}\mathbf{D}u(t)$

If $\frac{dt_0}{dt} \neq 0$, then as $N \rightarrow \infty$, from (E.8), the above comes out to

$$x_n^{(3)}(t) = - \left(\bar{c} \sum_{k=0}^{N-1} \mathbf{A}'_k \cdot x_k(t) + \bar{d}u(t) \right) \cdot (p_n \omega \psi)(\sigma(t, t_0)) (\sigma_s \phi)(t, t_0) \cdot \frac{dt_0}{dt}$$

It follows that in vectorized form we have $x^{(3)}(t) = -\bar{c}\mathbf{D}(\mathbf{A}')^\top x(t) - \bar{d}\mathbf{D}u(t)$. The result follows after combining the terms.

□

We see that the behavior of the model is dictated by t_0 . In particular, in this paper, we will consider two special cases.

Corollary E.4 (t_0 independent of t). *The SSM $((\mathbf{A} + c(t) + d(t)\mathbf{I}), \mathbf{B})$ satisfying conditions of Theorem E.3 with t_0 independent of t , is an O-SSM for basis functions $\chi p_n(\sigma)$ with measure $\omega \mathbb{I}[t_0, t] \sigma_s \chi^{-2}$ where $\mathbf{A} = \gamma_{nk}$ as in (E.6) and $\mathbf{B}_n = (p_n \omega \psi)(1)(\sigma_s \phi)(t, t)$.*

Proof. Follows from Theorem E.3. Since t_0 is independent of t , then $\frac{dt_0}{dt} = 0$, and $\mathbf{D} = \mathbf{0}$. □

Corollary E.5 ($t_0 = t - \theta$). *The SSM $(\mathbf{A}^0 + (c(t) + d(t))\mathbf{I} - \bar{c}\mathbf{D}\mathbf{A}', \mathbf{B} - \bar{d}\mathbf{D})$ satisfying conditions of Theorem E.3 with $t_0 = t - \theta$ for a fixed θ , is an O-SSM with basis functions $\chi p_n(\sigma)$ with measure $\omega \mathbb{I}[t_0, t] \sigma_s \chi^{-2}$ where $\mathbf{A}_{nk}^0 = \gamma_{nk}$ as in (E.6), $\mathbf{D}_n = (p_n \omega \psi)(\sigma(t, t - \theta))(\sigma_s \phi)(t, t - \theta)$, and $\mathbf{B}_n = (p_n \omega \psi)(1)(\sigma_s \phi)(t, t)$.*

Proof. This follows directly from Theorem E.3 by setting $t_0 = t - \theta$. □

E.3 Time-Varying Windows

E.3.1 Explanation of Time-Invariant LegS

Consider the case when

$$\sigma = \omega^{-1},$$

i.e. the measure is completely “tilted” away, and let

$$\frac{\partial}{\partial t} \sigma(t, s) = a(t) \sigma(t, s) + b(t). \quad (\text{E.11})$$

Let's consider the special case of (E.11) where $b(t) = 0$. This is most generally satisfied by

$$\sigma(t, s) = \exp(a(t) + z(s)).$$

Note that the condition $\sigma(t, t) = 1$ forces $z = -a$. Hence, we have

$$\sigma(t, s) = \exp(a(s) - a(t)). \quad (\text{E.12})$$

We now consider the following special case of Corollary E.4:

Corollary E.6. *Let $\eta \geq 0$. The SSM $(-a'(t)(\mathbf{A} + (\eta+1)\mathbf{I}), a'(t)\mathbf{B})$, where t_0 is independent of t , is an O-SSM for basis functions and measure*

$$\frac{\omega(\sigma)}{\sigma^\eta} p_n(\sigma) \quad \omega(t, s) = \mathbb{I}(\sigma) \frac{a'(s)\sigma^{2\eta+1}}{\omega(\sigma)}$$

where σ satisfies (E.12),

$$\phi(t, s) = \exp(\eta a(s) - \eta a(t)) = \sigma^\eta, \quad (\text{E.13})$$

$\mathbf{A} = \alpha_{nk}$ such that

$$y p'_n(y) = \sum_{k=0}^{n-1} \alpha_{nk} p_k(y) \quad (\text{E.14})$$

and

$$\mathbf{B}_n = p_n(1).$$

Proof. Given a orthonormal basis p_0, p_1, \dots, p_{N-1} with respect to a measure ω . Note that time-warping function σ satisfying (E.12) implies that $\sigma_s = a'(s)\sigma$.

We fix tilting $\chi(t, s) = \frac{\omega(\sigma)}{\sigma^\eta}$, which in turn follows by setting

$$\psi = \omega^{-1}.$$

We show shortly that we satisfy the pre-conditions of Corollary E.4, which implies (with our choice of χ and σ) that we have an O-SSM with basis functions $p_n(t, s) = \frac{\omega(\sigma)}{\sigma^\eta} p_n(\sigma)$ and measure

$$\begin{aligned}\omega(t, s) &= \omega(\sigma(t, s))\mathbb{I}[t_0, t]\sigma_s(t, s)\chi(t, s)^{-2} \\ &= \mathbb{I}[t_0, t]\frac{a'(s)\sigma^{2\eta+1}}{\omega(\sigma)}\end{aligned}$$

To complete the proof, we show that our choice of parameters above satisfies the conditions of Corollary E.4 (by showing they satisfy the conditions of Theorem E.3). We verify that σ and ϕ satisfy (E.3) and (E.7), noting that

$$\partial_t(\sigma_s) = -a'(t)\sigma_s,$$

and

$$\partial_t(\phi) = -\eta a'(t)\phi.$$

This implies that setting $c(t) = -a'(t)$ and $d(t) = -\eta a'(t)$ is enough to satisfy (E.3) and (E.7).

Further, note that (E.12) and the fact that $\psi = \omega^{-1}$ imply that

$$\sigma_t(p_n\omega\psi)'(\sigma) = -a'(t)\sigma p'_n(\sigma).$$

It follows that (E.6) is satisfied as long as

$$\sigma p'_n(\sigma) = \sum_{k=0}^{n-1} \alpha_{nk} p_k(\sigma)$$

for some set of coefficients $\{\alpha_{nk}\}_{k=0}^{N-1}$, which is exactly (E.14). This implies the γ_{nk} in Corollary E.4 satisfy.

$$\gamma_{nk} = -a'(t)\alpha_{nk}.$$

Let \mathbf{A} be the matrix such that $\mathbf{A}_{nk} = -\alpha_{nk}$ and then note that $-a'(t)(\mathbf{A} + (\eta + 1)\mathbf{I})$ is exactly the first parameter of the SSM in Corollary E.4. Similarly, recall in Corollary E.4

$$\begin{aligned} \mathbf{B}_n &= p_n(1)(\sigma_s\phi)(t, t) \\ &= p_n(1)a'(t), \end{aligned}$$

where the final equality follows since in our case, $\sigma_s(t, t) = a'(t) \exp(a(t) - a(t)) = a'(t)$. Overloading notation and letting $\mathbf{B}_n = p_n(1)$, all conditions of Corollary E.4 hold, from which the claimed result follows. \square

We are particularly interested in the following two special cases of Corollary E.6.

Corollary E.7. *The SSM $(-\frac{1}{t}(\mathbf{A} + \mathbf{I}), \frac{1}{t}\mathbf{B})$ is a O-SSM for basis functions $p_n(\frac{s}{t})\omega(\frac{s}{t})$ with measure $\frac{1}{t}\mathbb{I}[t_0, t] \left(\frac{s}{t}\right) \cdot \omega(\frac{s}{t})$ where $\mathbf{A} = \alpha_{nk}$ as in (E.14) and $\mathbf{B}_n = p_n(1)$.*

Proof. Letting $a'(t) = \frac{1}{t}$ implies that $a(t) = \ln t$. Then we can observe that is a case of

Corollary E.6 with time warping

$$\begin{aligned}\sigma(t, s) &= \exp(-\ln t + \ln s) \\ &= \exp(\ln(s/t)) \\ &= \frac{s}{t}.\end{aligned}$$

We set $\eta = 0$ in Corollary E.6, which in turn sets $\phi = \sigma^0 = 1$. This gives the tilting

$$\begin{aligned}\chi &= \phi^{-1}\psi^{-1} \\ &= \omega.\end{aligned}$$

Then by Corollary E.6, it follows that we can use σ and χ to build an O-SSM with basis functions

$$\frac{\omega(\sigma)}{\sigma^\eta} p_n(\sigma) = \omega\left(\frac{s}{t}\right) \cdot p_n\left(\frac{s}{t}\right)$$

with measure

$$\mathbb{I}(\sigma) \frac{a'(s)\sigma^{2\eta+1}}{\omega(\sigma)} = \frac{1}{t} \mathbb{I}(\sigma) \frac{\sigma}{\omega(\sigma)}.$$

Then the result follows. \square

Corollary E.8. *The SSM $(-(\mathbf{A} + \mathbf{I}), \mathbf{B})$ is a O-SSM for basis functions $p_n(e^{s-t})\omega(e^{s-t})$ with measure $\omega = \mathbb{I}[t_0, t](e^{s-t}) \frac{e^{s-t}}{\omega(e^{s-t})}$ where $\mathbf{A} = \alpha_{nk}$ as in (E.14) and $\mathbf{B}_n = p_n(1)$.*

Proof. This is a case of Corollary E.6 where $a'(t) = 1$, $\sigma = \exp(s - t)$, and we pick $\eta = 0$, implying that $\phi = \sigma^0 = 1$. It follows that

$$\begin{aligned}\chi &= \phi^{-1}\psi^{-1} \\ &= \omega.\end{aligned}$$

Utilizing Corollary E.6, we can use σ and χ to build an O-SSM with basis functions

$$\frac{\omega(\sigma)}{\sigma^\eta} p_n(\sigma) = \omega(\exp(s - t)) \cdot p_n(\exp(s - t))$$

with measure

$$\mathbb{I}(\sigma) \frac{a'(s)\sigma^{2\eta+1}}{\omega(\sigma)} = \mathbb{I}(\sigma) \frac{\exp(s-t)}{\omega(\exp(s-t))}.$$

This gives us our final result. \square

Next we instantiate Corollary E.6 to prove Corollary 5.8. (Even though strictly not needed, we instantiate Corollary E.8 and Corollary E.7 to prove Theorem 5.7 and Corollary 5.10.) To that end, we will need the following result:

Lemma E.9. *Let the Legendre polynomials orthonormal over the interval $[0, 1]$ be denoted as L_n . Then*

$$yL'_n(y) = nL_n(y) + \sqrt{2n+1} \left(\sum_{k=0}^{n-1} \sqrt{2k+1} L_k(y) \right), \quad (\text{E.15})$$

$$L'_n(y) = 2\sqrt{2n+1} \left(\sum_{0 \leq k \leq n-1, n-k \text{ is odd}} \sqrt{2k+1} L_k(y) \right), \quad (\text{E.16})$$

and

$$L_n(0) = (2n+1)^{\frac{1}{2}}(-1)^n \text{ and } L_n(1) = (2n+1)^{\frac{1}{2}}. \quad (\text{E.17})$$

Proof. The Legendre polynomials $P_n(t)$ satisfy the following orthogonality condition over $[-1, 1]$ (see Appendix A.2.1, equation (A.6)):

$$\int_{-1}^1 P_m(z)P_n(z) dz = \frac{2}{2n+1} \delta_{mn}.$$

Let us denote the normalized Legendre polynomials orthogonal over $[-1, 1]$ as $\lambda_n P_n(z)$ where $\lambda_n = \sqrt{\frac{2n+1}{2}}$. To orthogonalize them over $[0, 1]$, let $y = \frac{1+z}{2}$. It follows that $z = 2y - 1$, $dz = 2 dy$. Note that we then have

$$\int_{-1}^1 P_m(z)P_n(z) dz = \int_0^1 2P_m(2y-1)P_n(2y-1) dy.$$

This implies that

$$\int_{-1}^1 \frac{2n+1}{2} \cdot 2P_m(2y-1)P_n(2y-1) dy = \delta_{mn}.$$

Then if we let

$$L_n(y) = \sqrt{2}\lambda_n P_n(2y-1) = \sqrt{2n+1}P_n(2y-1), \quad (\text{E.18})$$

then we have an a set of functions over $[0, 1]$ such that

$$\int_0^1 L_m(y)L_n(y) dy = \delta_{mn}.$$

From [26, (2.8), (2.9)], note that $P_n(-1) = (-1)^n$ and $P_n(1) = 1$. This implies that

$$L_n(0) = \sqrt{2n+1}P_n(-1), \quad L_n(1) = \sqrt{2n+1}P_n(1).$$

Finally note that (E.18) implies:

$$\begin{aligned} L'_n(y) &= 2\sqrt{2n+1}P'_n(2y-1) \\ &= 2\sqrt{2n+1}P'_n(z). \end{aligned}$$

From [70, p. 7], we get

$$P'_n(z) = \sum_{\substack{0 \leq k \leq n-1, \\ n-k \text{ is odd}}} (2k+1)P_k(z).$$

Using (E.18) on the above, we get (E.16).

We now consider

$$\begin{aligned} yL'_n(y) &= 2y\sqrt{2n+1}P'_n(z) \\ &= (1+z)\sqrt{2n+1}P'_n(z). \end{aligned}$$

From [70, p. 8], we get

$$(z+1)P'_n(z) = nP_n(z) + \sum_{k=0}^{n-1} (2k+1)P_k(z).$$

Then the above becomes

$$yL'_n(y) = \sqrt{2n+1} \left(nP_n(z) + \sum_{k=0}^{n-1} (2k+1)P_k(z) \right).$$

(E.18) implies that $P_n(z) = \frac{L_n(y)}{\sqrt{2n+1}}$, thus

$$yL'_n(y) = nL_n(z) + \sqrt{2n+1} \left(\sum_{k=0}^{n-1} \sqrt{2k+1} L_k(z) \right).$$

□

We now re-state and prove Corollary 5.8:

Corollary E.10 (Corollary 5.8, restated). *Let L_n be the Legendre polynomials orthonormal over the interval $[0, 1]$. Define $\sigma(t, s) = \exp(a(s) - a(t))$. The SSM $(a'(t)\mathbf{A}, a'(t)\mathbf{B})$ is an O-SSM with*

$$\omega(t, s) = \mathbb{I}(\sigma(t, s))a'(s)\sigma(t, s) \quad p_n(t, s) = L_n(\sigma(t, s)),$$

where \mathbf{A} and \mathbf{B} are defined as in (5.1).

Proof. We consider our basis functions, the Legendre polynomials, which are orthogonal with respect to unit measure. This allows us to invoke Corollary E.6 with $\omega = 1$. Further, here we have $t_0 = -\infty$ and $\eta = 0$. Now we have an SSM:

$$(-a'(t)(\mathbf{A}^0 + \mathbf{I}), a'(t)\mathbf{B})$$

where $\mathbf{A}_{nk}^0 = \alpha_{nk}$ as in (E.14) and $\mathbf{B}_n = L_n(1)$.

From (E.17) observe that $\mathbf{B}_n = (2n+1)^{\frac{1}{2}}$. From (E.15), we have

$$\alpha_{nk} = \begin{cases} (2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} & k < n \\ n & k = n \\ 0 & \text{otherwise} \end{cases}.$$

We write that $\mathbf{A} = -(\mathbf{A}^0 + \mathbf{I})$. Indeed,

$$-(\mathbf{A}^0 + \mathbf{I})_{nk} = -\begin{cases} (2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} & \text{if } k < n \\ n+1 & \text{if } k = n \\ 0 & \text{if } k > n \end{cases}.$$

Thus the \mathbf{A} and \mathbf{B} match those in (5.1), which completes our claim. \square

We now re-state and prove Theorem 5.7:

Corollary E.11 (Theorem 5.7, restated). *Let L_n be the Legendre polynomials orthonormal over the interval $[0, 1]$. Then the SSM $(\frac{1}{t}\mathbf{A}, \frac{1}{t}\mathbf{B})$ is a O-SSM for basis functions $L_n(\frac{s}{t})$ and measure $\frac{1}{t}\mathbb{I}[t_0, t]$ where \mathbf{A} and \mathbf{B} are defined as in (5.1).*

Proof. We consider our basis functions, the Legendre polynomials, which are orthogonal with respect to unit measure. This allows us to invoke Corollary E.7 with $\omega = 1$. Now we have

$$x'(t) = \frac{1}{t} [-(\mathbf{A}^0 + \mathbf{I})x(t) + \mathbf{B}u(t)]$$

where $\mathbf{A}_{nk}^0 = \alpha_{nk}$ as in (E.14) and $\mathbf{B}_n = L_n(1)$.

From (E.17) observe that $\mathbf{B}_n = (2n+1)^{\frac{1}{2}}$. From (E.15), we have

$$\alpha_{nk} = \begin{cases} (2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} & k < n \\ n & k = n \\ 0 & \text{otherwise} \end{cases}.$$

We write that $\mathbf{A} = -(\mathbf{A}^0 + \mathbf{I})$. Indeed,

$$-(\mathbf{A}^0 + \mathbf{I})_{nk} = - \begin{cases} (2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} & \text{if } k < n \\ n+1 & \text{if } k = n \\ 0 & \text{if } k > n \end{cases}$$

which completes our claim. \square

We now restate and prove Corollary 5.10.

Corollary E.12 (Corollary 5.10, restated). *Let L_n be the Legendre polynomials orthonormal over the interval $[0, 1]$. Then the SSM (\mathbf{A}, \mathbf{B}) is a TO-SSM for basis functions $L_n(e^{-t})$ with measure $\omega = \mathbb{I}[t_0, t]e^{-t}$ where \mathbf{A}, \mathbf{B} are defined as in (5.1).*

Proof. We consider our basis functions, the Legendre polynomials, which are orthogonal with respect to unit measure, warping function $\sigma = \exp(s-t)$, and with tilting $\chi = \omega$. We note that $\sigma = \exp(s-t)$ satisfies (E.12) with, $a'(t) = 1$. This allows us to invoke Corollary E.7.

Then $x'(t) = (\mathbf{A} + \mathbf{I})x(t) + \mathbf{B}u(t)$ orthogonalizes against the basis functions $L_n(e^{s-t})$ with measure $\mathbb{I}[-\infty, t]e^{s-t}$ where $\mathbf{A} = \alpha_{nk}$ as in E.14. Note that the SSM basis functions $K_n(t, s) = K_n(s-t)$, hence we get the claimed SSM form utilizing the same argument for \mathbf{A}, \mathbf{B} as in the proof of Corollary E.11 \square

This explains why removing the $\frac{1}{t}$ factor from HIPPO-LegS still works: it is orthogonalizing onto the Legendre polynomials with an exponential “warping”.

E.4 Finite Windows

E.4.1 Derivation of LegT

Corollary E.13. *Let L_n be the Legendre polynomials orthonormal over the interval $[0, 1]$ and let $\sigma = 1 - \frac{t-s}{\theta}$ for a constant θ . Then the SSM $(\frac{1}{\theta}\mathbf{A}, \frac{1}{\theta}\mathbf{B})$ is a O-SSM for basis functions $L_n(\sigma)$ with measure $\frac{1}{\theta}\mathbb{I}[t_0, t](\sigma)$ where \mathbf{A}, \mathbf{B} are defined as in (5.2).*

Proof. Our plan is to apply Corollary E.5, for which we must show that the basis functions $L_n(t, s)$, time warping $\sigma(t, s)$, and tilting $\chi(t, s) = \psi^{-1}\phi^{-1}(t, s)$ satisfy (E.6), (E.3), and (E.7), respectively. We first set some parameters—note that because $\omega = 1$ and set $\psi = \phi = 1$.

The above implies that we have

$$\sigma_t(L_n\omega\psi)'(\sigma) = -\frac{1}{\theta}L'_n(\sigma).$$

The above along with (E.16), we see that the Legendre polynomials satisfy (E.6) with

$$\gamma_{nk} = \frac{1}{\theta} \cdot \begin{cases} -2 \cdot (2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} & k < n \text{ and } n-k \text{ is odd} \\ 0 & \text{otherwise} \end{cases}. \quad (\text{E.19})$$

We also note that $\sigma_s = \frac{1}{\theta}$.

It follows that

$$\frac{d}{dt}\sigma_s = 0,$$

satisfying (E.3) trivially by setting $c(t) = 0$. Similarly, since $\phi = 1$ (E.7) is also satisfied trivially by setting $d(t) = 0$. Finally we note that the L_n forms a complete basis over $[0, 1]$, hence as $N \rightarrow \infty$, we have

$$u(t - \theta) = \sum_{k=0}^{N-1} x_k(t)L_n(\sigma(t, t - \theta)) = \sum_{k=0}^{N-1} x_k(t)L_n(0).$$

The above defines \mathbf{A}' by setting $\mathbf{A}'_n = L_n(0)$ (as well as $\bar{c} = 1$ and $\bar{d} = 0$.) Now by Corollary E.5, we have an SSM

$$\left(\mathbf{A}^0 - \mathbf{D} (\mathbf{A}')^\top, \mathbf{B}' \right),$$

where $\mathbf{D}_n = \frac{1}{\theta} L_n(0)$, and by (E.15) $\mathbf{A}_{nk}^0 = \gamma_{nk}$ (as in (E.19)) and $\mathbf{B}'_n = \frac{1}{\theta} L_n(1)$.

From (E.17), we have $\mathbf{D}_n = \frac{1}{\theta} (2n+1)^{\frac{1}{2}} (-1)^n$ and $\mathbf{B}_n = \frac{1}{\theta} (2n+1)^{\frac{1}{2}}$.

Thus, we have

$$\left(\mathbf{A}^0 - \mathbf{D} (\mathbf{A}')^\top \right)_{nk} = \frac{1}{\theta} \cdot \begin{cases} -(2n+1)^{\frac{1}{2}} (2k+1)^{\frac{1}{2}} (2 + (-1)^{n-k}) & k < n \text{ and } n - k \text{ is odd} \\ -(2n+1)^{\frac{1}{2}} (2k+1)^{\frac{1}{2}} (-1)^{n-k} & \text{otherwise} \end{cases}.$$

The proof is complete by noting that $\mathbf{A}^0 - \mathbf{D} (\mathbf{A}')^\top = \frac{1}{\theta} \mathbf{A}$ and $\mathbf{B}' = \frac{1}{\theta} \mathbf{B}$. \square

We note that Corollary E.13 implies Proposition 5.6. More specifically, Proposition 5.6 follows by setting $\theta = 1$ in Corollary E.13 and noticing that the O-SSM there is actually a TO-SSM. (Technically we get basis function $L_n(1-t)$ for measure $\mathbb{I}(1-t)$ but this is fine since $\int_0^1 L_k(1-t) L_j(1-t) dt = \int_0^1 L_k(t) L_j(t) dt$.)

We first give a proof of Theorem 5.11. Then, we prove Theorem 5.12 as a function approximation result pertaining to S4-FouT.

E.4.2 Explanation of FouT

Proof of Theorem 5.11. We seek to derive \mathbf{A} and \mathbf{B}' from (5.3) using Corollary E.5:

We use the time-warping function $\sigma(t, s) = 1 - (t - s)$, which implies that we have

$$\sigma_s(t, s) = 1, \tag{E.20}$$

$$\frac{\partial}{\partial t} \sigma_s(t, s) = 0 \tag{E.21}$$

Thus, we can take

$$c(t) = 0 \text{ in } \frac{\partial}{\partial t} \sigma_s(t, s) = c(t) \sigma_s(t, s). \tag{E.22}$$

We then have $\chi(t, s) = 1$ as we set

$$\psi(t, s) = \phi(t, s) = 1, \quad (\text{E.23})$$

$$\frac{d}{dt}\phi(t, s) = 0. \quad (\text{E.24})$$

So, we can take

$$d(t) = 0 \text{ in } \frac{d}{dt}\phi(t, s) = d(t)\phi(t, s). \quad (\text{E.25})$$

We also have $\omega(\sigma) = 1$, and we order our bases in the form $p_n = (1, c_1(t), s_1(t), c_2(t), s_2(t), \dots)^1$, where the basis functions have derivatives:

$$\begin{aligned} (1)'(\sigma) &= 0; \\ (c_n)'(\sigma) &= -2\pi n s_n(\sigma); \\ (s_n)'(\sigma) &= 2\pi n c_n(\sigma). \end{aligned}$$

Consequently, we can define γ_{nk} as follows:

$$\gamma_{nk} = \begin{cases} 2\pi n & n - k = 1, \text{ } k \text{ odd} \\ -2\pi k & k - n = 1, \text{ } n \text{ odd} \\ 0 & \text{otherwise} \end{cases}. \quad (\text{E.26})$$

Further, the discontinuity is at $t_0 = t - \theta$, $\theta = 1$ which implies that $\frac{dt_0}{dt} = 1$. We now seek to use the stored approximation to u at time t to compute $u(t - 1)$.

First, denote the latent state $x(t)$ with coefficients $x = (x^1(t), x_1^c(t), x_1^s(t), x_2^c(t), x_2^s(t), \dots)$ and define the functions $v(s)$ and $w(s)$ such that we have

$$v(s) = u(2t - s - 1) \quad \text{and} \quad w(s) = \frac{u(s) + v(s)}{2} \quad \text{for } s \in [t - 1, t].$$

¹Note that this is 0-indexed.

Now, let \hat{u} , \hat{v} , and \hat{w} denote the reconstruction of u , v and w , where we have

$$\hat{u}(t, s) = \langle u(s), 1 \rangle + \sum_n \langle u(s), s_n(\sigma(t, s)) \rangle s_n(\sigma(t, s)) + \sum_n \langle u(s), c_n(\sigma(t, s)) \rangle c_n(\sigma(t, s)), \quad (\text{E.27})$$

$$\hat{v}(t, s) = \langle v(s), 1 \rangle + \sum_n \langle v(s), s_n(\sigma(t, s)) \rangle s_n(\sigma(t, s)) + \sum_n \langle v(s), c_n(\sigma(t, s)) \rangle c_n(\sigma(t, s)), \quad (\text{E.28})$$

$$\hat{w}(t, s) = \langle w(s), 1 \rangle + \sum_n \langle w(s), s_n(\sigma(t, s)) \rangle s_n(\sigma(t, s)) + \sum_n \langle w(s), c_n(\sigma(t, s)) \rangle c_n(\sigma(t, s)). \quad (\text{E.29})$$

Then, we claim that

$$\hat{u}(t, t) = \frac{u(t) + v(t)}{2} = \frac{u(t) + u(t-1)}{2}. \quad (\text{E.30})$$

Towards that end, we examine the sine and cosine coefficients of u and v as follows:

$$\begin{aligned} \langle v, c_n \rangle &= \int v(s) c_n(\sigma(t, s)) \mathbb{I}[t-1, t] ds = \int u(2t-s-1) c_n(\sigma(t, s)) \mathbb{I}[t-1, t] ds \\ &= \int u(s') c_n(1 - \sigma(t, s')) \mathbb{I}[t-1, t] ds' \\ &= \int u(s') c_n(\sigma(t, s')) \mathbb{I}[t-1, t] ds' = \langle u, c_n \rangle. \end{aligned} \quad (\text{E.31})$$

$$\begin{aligned} \langle v, s_n \rangle &= \int v(s) s_n(\sigma(t, s)) \mathbb{I}[t-1, t] ds = \int u(2t-s-1) s_n(\sigma(t, s)) \mathbb{I}[t-1, t] ds \\ &= \int u(s') s_n(1 - \sigma(t, s')) \mathbb{I}[t-1, t] ds' \\ &= - \int u(s') s_n(\sigma(t, s')) \mathbb{I}[t-1, t] ds' = -\langle u, s_n \rangle. \end{aligned} \quad (\text{E.32})$$

Here, for (E.31) and (E.32), we use the change of variables $s' \leftarrow 2t-s-1$, which gives us

$$\sigma(t, s) = 1 - (t - s) = 1 - (1 + t - s - 1) = 1 - [1 - (t - (2t - s - 1))] = 1 - (1 - (t - s')) = 1 - \sigma(t, s').$$

Then, we use the fact that $c_n(1 - \sigma(t, s')) = c_n(\sigma(t, s'))$ but $s_n(1 - \sigma(t, s')) = -s_n(\sigma(t, s'))$. That is, both u and v have the same cosine coefficients but negated sine coefficients of each other. But, we know that both $s_n(\sigma(t, t-1)) = s_n(1 - (t - (t-1))) = s_n(0) = 0$ and $s_n(\sigma(t, t)) = s_n(1 - (t - t)) = s_n(1) = 0$, and hence, the reconstruction of \hat{u} at the endpoints $\sigma(t, t-1) = 0$ and $\sigma(t, t) = 1$ depends only on the cosine coefficients, whence we assert that

the reconstruction \hat{u} agrees with \hat{v} at both endpoints. Therefore, we have $\hat{u}(t, t) = \hat{v}(t, t)$ implying that $\hat{w}(t, t) = \hat{u}(t, t)$.

Note that w is continuous and periodic, for which the basis $\{1, c_n, s_n\}_n$ is complete, and hence, we know that as $N \rightarrow \infty$, $\hat{w} \rightarrow w$. Thus, at $s = t$, we have $\hat{u}(t, t) = \hat{w}(t, t) = w(t) = \frac{u(t)+v(t)}{2} = \frac{u(t)+u(t-1)}{2}$, which completes the proof of the claim in (E.30).

Recall from (E.27) that we can express the stored approximation of $u(t)$, given by $\hat{u}(t, s)$, as follows:

$$\hat{u}(t, s) = \langle u(s), 1 \rangle + \sum_n \langle u(s), s_n(\sigma(t, s)) \rangle s_n(\sigma(t, s)) + \sum_n \langle u(s), c_n(\sigma(t, s)) \rangle c_n(\sigma(t, s))$$

For the value at t , the approximation $\hat{u}(t, t)$ is then given by

$$\hat{u}(t, t) = x^1(t) + \sum_k x_k^c(t) c_k(1) + \sum_k x_k^s(t) s_k(1) = x^1(t) + \sum_k \sqrt{2} x_k^c(t).$$

Due to (E.30), we know $u(t-1) = 2\hat{u}(t, t) - u(t)$, which combined with the above yields:

$$u(t-1) = 2x^1(t) + 2\sqrt{2} \sum_k x_k^c(t) - u(t). \quad (\text{E.33})$$

Finally, with regards to Corollary E.5, for Theorem E.3, (E.22) satisfies (E.3) and (E.25) satisfies (E.7) with (E.26) satisfying (E.6) for \mathbf{A}^0 . Moreover, from (E.33), we can take

$$\bar{c} = 1, \bar{d} = -1, \text{ and } \mathbf{A}'_k := \begin{cases} 2 & k = 0 \\ 2\sqrt{2} & k \text{ odd} \\ 0 & \text{otherwise} \end{cases} \quad \text{to satisfy (E.8).}$$

Invoking Corollary E.5 now yields the following O-SSM:²

$$(\mathbf{A}^0 + (c(t) + d(t))\mathbf{I} - \bar{c}\mathbf{D} (\mathbf{A}')^\top, \mathbf{B} - \bar{d}\mathbf{D}),$$

²Recall that, like the coefficients, the matrices are 0-indexed.

where $\mathbf{A}_{nk}^0 = \gamma_{nk}$ with \mathbf{D}_n and \mathbf{B}_n specified as follows:

$$\mathbf{D}_n = \begin{cases} 1 & n = 0 \\ \sqrt{2} & n \text{ odd} \\ 0 & \text{otherwise} \end{cases} \quad (\text{E.34})$$

$$\mathbf{B}_n = \begin{cases} 1 & n = 0 \\ \sqrt{2}, & n \text{ odd} \\ 0 & \text{otherwise} \end{cases} \quad (\text{E.35})$$

Here, the values are derived from the expressions of Corollary E.5:

$$\mathbf{D}_n = (p_n \omega \psi)(\sigma(t, t-1))(\sigma_s \phi)(t, t-1) \text{ and } \mathbf{B}_n = (p_n \omega \psi)(1)(\sigma_s \phi)(t, t).$$

Recall that we have $p_n \in \{1, c_n, s_n\}$, $\omega(t, s) = 1$, and from (E.20) and (E.23), $\sigma_s(t, s) = 1$ with $\psi(t, s) = \phi(t, s) = 1$. Thus, (E.34) is due to $1(0) \cdot 1 = 1$, $s_n(0) \cdot 1 = 0$ but $c_n(0) \cdot 1 = \sqrt{2}$. Similarly, (E.35) is because $1(0) \cdot 1 = 1$, $s_n(1) \cdot 1 = 0$ but again $c_n(1) \cdot 1 = \sqrt{2}$.

Now, we have

$$[\mathbf{D}(\mathbf{A}')^\top]_{nk} = \begin{cases} 2 & n = k = 0 \\ 2\sqrt{2} & n = 0, k \text{ odd or } k = 0, n \text{ odd} \\ 4 & n, k \text{ odd} \\ 0 & \text{otherwise} \end{cases} \quad [\bar{\mathbf{D}}]_n = \begin{cases} -1 & n = 0 \\ -\sqrt{2} & n \text{ odd} \\ 0 & n \text{ otherwise} \end{cases}.$$

As $c(t) = d(t) = 0$, we define $\mathbf{A} \leftarrow \mathbf{A}^0 - \bar{c}\mathbf{D}(\mathbf{A}')^\top$ and $\mathbf{B} \leftarrow \mathbf{B} - \bar{d}\mathbf{D}$, given by

$$\mathbf{A}_{nk} = \begin{cases} -2 & n = k = 0 \\ -2\sqrt{2} & n = 0, k \text{ odd or } k = 0, n \text{ odd} \\ -4 & n, k \text{ odd} \\ 2\pi n & n - k = 1, k \text{ odd} \\ -2\pi k & k - n = 1, n \text{ odd} \\ 0 & \text{otherwise} \end{cases} \quad \mathbf{B}_n = \begin{cases} 2 & n = 0 \\ 2\sqrt{2} & n \text{ odd} \\ 0 & \text{otherwise} \end{cases}$$

□

E.4.3 Function Approximation Error

Proof of Theorem 5.12. First, the state size being N dictates that there are $\lfloor N/2 \rfloor$ s_n and c_n basis functions each. We fix time t and denote x_n^c and x_n^s to be the respective coefficients for s_n and c_n basis corresponding to S4-Fou. Since $\{s_n, c_n\}_{n \geq 0}$ forms an orthonormal basis, by Parseval's identity, we have

$$\|K - \hat{K}\|_2^2 = \sum_{n=\lfloor N/2 \rfloor}^{\infty} |x_n^c|^2(t) + |x_n^s|^2(t). \quad (\text{E.36})$$

Thus, in order to bound the error, it suffices to bound the high-order coefficients by integration by parts as follows:

$$\begin{aligned} x_n^c(t) &= \langle K, c_n \rangle = \int_0^1 K(t) c_n(t) dt \\ &= \left[K(t) \frac{1}{2\pi n} s_n(t) \right]_0^1 - \frac{1}{2\pi n} \int_0^1 K'(t) s_n(t) dt \\ &= -\frac{1}{2\pi n} \int_0^1 K'(t) s_n(t) dt. \end{aligned}$$

The quantity in the bracket vanishes as s_n is periodic. Therefore

$$|x_n^c| \leq \left| \frac{1}{2\pi n} \int_0^1 K'(t) s_n(t) dt \right| \leq \frac{1}{2\pi n} \int_0^1 |K'(t)| |s_n(t)| dt \leq \frac{L}{2\pi n},$$

where we use the fact that K is L -Lipshitz. For x_n^s , a similar argument holds and we get:

$$|x_n^s| \leq \left| \frac{1}{2\pi} \int_0^1 K'(t) c_n(t) dt \right| \leq \frac{1}{2\pi} \int_0^1 |K'(t)| |c_n(t)| dt \leq \frac{L}{2\pi n}.$$

Due to (E.36), this then implies that

$$\begin{aligned} \|K - \hat{K}\|_2^2 &= \sum_{n=\lfloor N/2 \rfloor}^{\infty} |x_n^c|^2(t) + |x_n^s|^2(t) = \sum_{n=\lfloor N/2 \rfloor}^{\infty} |x_n^c|^2(t) + |x_n^s|^2(t) \\ &\leq \sum_{n=\lfloor N/2 \rfloor}^{\infty} \frac{2L^2}{(2\pi n)^2} = \frac{2L^2}{(2\pi)^2} \sum_{n=\lfloor N/2 \rfloor}^{\infty} \frac{1}{n^2} = \frac{2L^2}{(2\pi)^2} \frac{1}{\lfloor N/2 \rfloor} \\ &\leq \frac{L^2}{\pi^2(N-2)}. \end{aligned} \quad (\text{E.37})$$

We use (E.37) to get the following estimate on $\|K - \hat{K}\|$:

$$\|K - \hat{K}\|_2 \leq \frac{L}{\pi\sqrt{(N-2)}}.$$

Thus, it suffices for N to satisfy the following inequality:

$$\frac{L}{\pi\sqrt{(N-2)}} \leq \epsilon \implies \sqrt{N-2} \geq \frac{L}{\pi\epsilon} \implies N \geq \left(\frac{L}{\pi\epsilon}\right)^2 + 2.$$

We now use the same argument as above to the fact that K has order- k bounded derivative. By iteration, we get:

$$|x_n^s| = |x_n^c| \leq \left| \frac{1}{(2\pi n)^k} \int_0^1 K^{(k)}(t) s_n(t) dt \right| \leq \frac{1}{(2\pi n)^k} \int_0^1 |K^{(k)}| |s_n(t)| dt \leq \frac{L}{(2\pi n)^k}.$$

Again, due to (E.36), this then gives us the following estimate on the square error:

$$\begin{aligned} \|K - \hat{K}\|_2^2 &= \sum_{n=\lfloor N/2 \rfloor}^{\infty} |x_n^c|^2(t) + |x_n^s|^2(t) = \sum_{n=\lfloor N/2 \rfloor}^{\infty} |x_n^c|^2(t) + |x_n^s|^2(t) \\ &\leq \sum_{n=\lfloor N/2 \rfloor}^{\infty} \frac{2L^2}{(2\pi n)^{2k}} = \frac{2L^2}{(2\pi)^{2k}} \sum_{n=\lfloor N/2 \rfloor}^{\infty} \frac{1}{n^{2k}} = \frac{2L^2}{(2\pi)^{2k}} \frac{1}{(\lfloor N/2 \rfloor)^{2k-1}} \\ &\leq \frac{L^2}{\pi^{2k}(N-2)^{2k-1}}. \end{aligned} \tag{E.38}$$

If K has order k -bounded derivatives, then we use (E.38) to get the following estimate on $\|K - \hat{K}\|$:

$$\|K - \hat{K}\|_2 \leq \frac{L}{\pi^k(N-2)^{-k+1/2}}.$$

Again, it suffices for N to satisfy the following inequality:

$$\frac{L}{\pi^k(N-2)^{-k+1/2}} \leq \epsilon \implies (N-2)^{k-1/2} \geq \frac{L}{\pi^k\epsilon} \implies N \geq \left(\frac{L}{\pi^k\epsilon}\right)^{\frac{2}{2k-1}} + 2.$$

□

E.4.4 Delay Network

Finally, we prove Theorem 5.14. Note that this is a stronger version of the LegT portion of Theorem 5.13, while the FouT portion is a corollary of the proof of Theorem 5.11.

We start by working out some calculations concretely to provide an example. The SSM corresponding to HIPPO-LegT is

$$\begin{aligned}\mathbf{A} &= \mathbf{P}^{\frac{1}{2}} \begin{bmatrix} -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 \end{bmatrix} \mathbf{P}^{\frac{1}{2}} \\ \mathbf{B} &= \mathbf{P}^{\frac{1}{2}} \mathbf{1} \\ \mathbf{C} &= \mathbf{Z}^\top \mathbf{P}^{\frac{1}{2}}\end{aligned}$$

$$\begin{aligned}\mathbf{P} &= \text{diag}\{1 + 2n\} \\ \mathbf{Z}^\top &= \begin{bmatrix} 1 & -1 & 1 & -1 \end{bmatrix}\end{aligned}$$

The transfer function is

$$\mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1} \mathbf{B} = \mathbf{Z}(s\mathbf{P}^{-1} - \mathbf{A})^{-1} \mathbf{1}$$

(In the RHS and for the rest of this part, we will redefine \mathbf{A} to be the ± 1 matrix found above for convenience.)

Case N=1

We have $\mathbf{A} = -1$, $\mathbf{B} = \mathbf{C} = 1$, and the transfer function is $\mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1} \mathbf{B} = \frac{1}{1+s}$.

Case N=2

The transfer function is

$$\begin{aligned}
 C(sI - A)^{-1}B &= \begin{bmatrix} 1 & -1 \end{bmatrix} \left(sP^{-1} - \begin{bmatrix} -1 & 1 \\ -1 & -1 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\
 &= \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} s+1 & -1 \\ 1 & \frac{s}{3} + 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\
 &= \frac{1}{\frac{s^2}{3} + \frac{4s}{3} + 2} \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 1 + \frac{s}{3} & 1 \\ -1 & 1 + s \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\
 &= \frac{2 - \frac{2s}{3}}{\frac{s^2}{3} + \frac{4s}{3} + 2} \\
 &= \frac{1 - \frac{s}{3}}{1 + \frac{2s}{3} + \frac{s^2}{6}}
 \end{aligned}$$

It can be verified that this is indeed $[1/2]_{\text{exp}}(-s)$.

A General Recursion

We will now sketch out a method to relate these transfer functions recursively.

We will redefine Z to be the vector that ENDS in +1.

The main idea is to write

$$\begin{aligned}
 A_n &= \begin{bmatrix} A_{n-1} & Z_{n-1} \\ -1^\top_{n-1} & -1 \end{bmatrix} \\
 (sP_n^{-1} - A_n)^{-1} &= \begin{bmatrix} sP_{n-1}^{-1} - A_{n-1} & -Z_{n-1} \\ 1_{n-1}^\top & 1 + \frac{s}{2n+1} \end{bmatrix}^{-1}.
 \end{aligned}$$

Now we can use the block matrix inversion formula.³ Ideally, this will produce a recurrence where the desired transfer function $Z_n(sP_n^{-1} - A_n)^{-1}1_n$ will depend on $Z_{n-1}(sP_n^{-1} - A_{n-1})^{-1}1_{n-1}$. However, looking at the block matrix inversion formula, it becomes clear that there are also dependencies on terms like $1_{n-1}^\top(sP_{n-1}^{-1} - A_{n-1})^{-1}1_{n-1}$ and $Z_{n-1}(sP_{n-1}^{-1} - A_{n-1})^{-1}Z_{n-1}^\top$.

The solution is to track all of these terms simultaneously.

³https://en.wikipedia.org/wiki/Block_matrix#/Block_matrix_inversion

We will compute the 4 transfer functions

$$\begin{aligned}\mathbf{H}_n(s) &:= \begin{bmatrix} \mathbf{H}_n^{1z}(s) & \mathbf{H}_n^{11}(s) \\ \mathbf{H}_n^{zz}(s) & \mathbf{H}_n^{z1}(s) \end{bmatrix} \\ &:= \begin{bmatrix} \mathbf{1}_n^\top (s\mathbf{P}_n^{-1} - \mathbf{A}_n)^{-1} \mathbf{Z}_n & \mathbf{1}_n^\top (s\mathbf{P}_n^{-1} - \mathbf{A}_n)^{-1} \mathbf{1}_n \\ \mathbf{Z}_n^\top (s\mathbf{P}_n^{-1} - \mathbf{A}_n)^{-1} \mathbf{Z}_n & \mathbf{Z}_n^\top (s\mathbf{P}_n^{-1} - \mathbf{A}_n)^{-1} \mathbf{1}_n \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{1}_n^\top \\ \mathbf{Z}_n^\top \end{bmatrix} (s\mathbf{P}_n^{-1} - \mathbf{A}_n)^{-1} \begin{bmatrix} \mathbf{Z}_n & \mathbf{1}_n \end{bmatrix}\end{aligned}$$

Lemma E.14. *Instead of using the explicit block matrix inversion formula, it will be easier to work with the following factorization used to derive it (block LDU decomposition⁴).*

$$\begin{aligned}\begin{bmatrix} A & B \\ C & D \end{bmatrix} &= \begin{bmatrix} I & 0 \\ CA^{-1} & I \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & D - CA^{-1}B \end{bmatrix} \begin{bmatrix} I & A^{-1}B \\ 0 & I \end{bmatrix} \\ \begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} &= \begin{bmatrix} I & -A^{-1}B \\ 0 & I \end{bmatrix} \begin{bmatrix} A^{-1} & 0 \\ 0 & (D - CA^{-1}B)^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -CA^{-1} & I \end{bmatrix}\end{aligned}$$

Using Lemma E.14, we can factor the inverse as

$$\begin{aligned}(s\mathbf{P}_n^{-1} - \mathbf{A}_n)^{-1} &= \begin{bmatrix} \mathbf{I}_{n-1} & (s\mathbf{P}_{n-1}^{-1} - \mathbf{A}_{n-1})^{-1} \mathbf{Z}_{n-1} \\ & 1 \end{bmatrix} \\ &\quad \begin{bmatrix} (s\mathbf{P}_{n-1}^{-1} - \mathbf{A}_{n-1})^{-1} & \\ & \left(1 + \frac{s}{2n+1} + (-1)^{n-1} \mathbf{H}_{n-1}^{1z}(s)\right)^{-1} \end{bmatrix} \\ &\quad \begin{bmatrix} \mathbf{I}_{n-1} & \\ -\mathbf{1}_{n-1}^\top (s\mathbf{P}_{n-1}^{-1} - \mathbf{A}_{n-1})^{-1} & 1 \end{bmatrix}\end{aligned}$$

⁴https://en.wikipedia.org/wiki/Schur_complement#/Background

Now we compute

$$\begin{aligned} & \begin{bmatrix} \mathbf{1}_n^\top \\ \mathbf{Z}_n^\top \end{bmatrix} \begin{bmatrix} \mathbf{I}_{n-1} & (s\mathbf{P}_{n-1}^{-1} - \mathbf{A}_{n-1})^{-1} \mathbf{Z}_{n-1} \\ & 1 \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{1}_{n-1}^\top & 1 \\ -\mathbf{Z}_{n-1}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I}_{n-1} & (s\mathbf{P}_{n-1}^{-1} - \mathbf{A}_{n-1})^{-1} \mathbf{Z}_{n-1} \\ & 1 \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{1}_{n-1}^\top & 1 + \mathbf{H}_{n-1}^{1z}(s) \\ -\mathbf{Z}_{n-1}^\top & 1 - \mathbf{H}_{n-1}^{zz}(s) \end{bmatrix} \end{aligned}$$

and

$$\begin{aligned} & \begin{bmatrix} \mathbf{I}_{n-1} \\ -\mathbf{1}_{n-1}^\top (s\mathbf{P}_{n-1}^{-1} - \mathbf{A}_{n-1})^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{Z}_n & \mathbf{1}_n \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{I}_{n-1} \\ -\mathbf{1}_{n-1}^\top (s\mathbf{P}_{n-1}^{-1} - \mathbf{A}_{n-1})^{-1} \end{bmatrix} \begin{bmatrix} -\mathbf{Z}_{n-1} & \mathbf{1}_{n-1} \\ 1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} -\mathbf{Z}_{n-1} & \mathbf{1}_{n-1} \\ 1 + \mathbf{H}_{n-1}^{1z}(s) & 1 - \mathbf{H}_{n-1}^{11}(s) \end{bmatrix} \end{aligned}$$

Now we can derive the full recurrence for all these functions.

$$\begin{aligned}
\mathbf{H}_n(s) &= \begin{bmatrix} \mathbf{H}_n^{1z}(s) & \mathbf{H}_n^{11}(s) \\ \mathbf{H}_n^{zz}(s) & \mathbf{H}_n^{z1}(s) \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{1}_n^\top \\ \mathbf{Z}_n^\top \end{bmatrix} (s\mathbf{P}_n^{-1} - \mathbf{A}_n)^{-1} \begin{bmatrix} \mathbf{Z}_n & \mathbf{1}_n \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{1}_n^\top \\ \mathbf{Z}_n^\top \end{bmatrix} \begin{bmatrix} \mathbf{I}_{n-1} & (s\mathbf{P}_{n-1}^{-1} - \mathbf{A}_{n-1})^{-1} \mathbf{Z}_{n-1} \\ & 1 \end{bmatrix} \\
&\quad \left[(s\mathbf{P}_{n-1}^{-1} - \mathbf{A}_{n-1})^{-1} \right. \\
&\quad \left. \left(1 + \frac{s}{2n+1} + (-1)^{n-1} \mathbf{H}_{n-1}^{1z}(s) \right)^{-1} \right] \\
&\quad \begin{bmatrix} \mathbf{I}_{n-1} \\ -\mathbf{1}_{n-1}^\top (s\mathbf{P}_{n-1}^{-1} - \mathbf{A}_{n-1})^{-1} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{Z}_n & \mathbf{1}_n \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{1}_{n-1}^\top & 1 + \mathbf{H}_{n-1}^{1z}(s) \\ -\mathbf{Z}_{n-1}^\top & 1 - \mathbf{H}_{n-1}^{zz}(s) \end{bmatrix} \\
&\quad \cdot \begin{bmatrix} (s\mathbf{P}_{n-1}^{-1} - \mathbf{A}_{n-1})^{-1} & \\ & \left(1 + \frac{s}{2n+1} + \mathbf{H}_{n-1}^{1z}(s) \right)^{-1} \end{bmatrix} \\
&\quad \cdot \begin{bmatrix} -\mathbf{Z}_{n-1} & \mathbf{1}_{n-1} \\ 1 + \mathbf{H}_{n-1}^{1z}(s) & 1 - \mathbf{H}_{n-1}^{11}(s) \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{1}_{n-1}^\top \\ -\mathbf{Z}_{n-1}^\top \end{bmatrix} (s\mathbf{P}_{n-1}^{-1} - \mathbf{A}_{n-1})^{-1} \begin{bmatrix} -\mathbf{Z}_{n-1} & \mathbf{1}_{n-1} \end{bmatrix} \\
&\quad + \begin{bmatrix} 1 + \mathbf{H}_{n-1}^{1z}(s) \\ 1 - \mathbf{H}_{n-1}^{zz}(s) \end{bmatrix} \left(1 + \frac{s}{2n+1} + \mathbf{H}_{n-1}^{1z}(s) \right)^{-1} \begin{bmatrix} 1 + \mathbf{H}_{n-1}^{1z}(s) & 1 - \mathbf{H}_{n-1}^{11}(s) \end{bmatrix} \\
&= \begin{bmatrix} -\mathbf{H}_{n-1}^{1z}(s) & \mathbf{H}_{n-1}^{11}(s) \\ \mathbf{H}_{n-1}^{zz}(s) & -\mathbf{H}_{n-1}^{z1}(s) \end{bmatrix} \\
&\quad + \begin{bmatrix} 1 + \mathbf{H}_{n-1}^{1z}(s) \\ 1 - \mathbf{H}_{n-1}^{zz}(s) \end{bmatrix} \left(1 + \frac{s}{2n+1} + \mathbf{H}_{n-1}^{1z}(s) \right)^{-1} \begin{bmatrix} 1 + \mathbf{H}_{n-1}^{1z}(s) & 1 - \mathbf{H}_{n-1}^{11}(s) \end{bmatrix}
\end{aligned}$$

Now we'll define a few transformations which will simplify the calculations. Define

$$\begin{aligned} G_n^{1z}(s) &= \frac{1}{2}(1 + \mathbf{H}_n^{1z}(s)) \\ G_n^{11}(s) &= 1 - \mathbf{H}_n^{1z}(s) \\ G_n^{zz}(s) &= 1 - \mathbf{H}_n^{1z}(s) \\ G_n^{z1}(s) &= (-1)^n \mathbf{H}_n^{z1}(s) \end{aligned}$$

These satisfy the following recurrences:

$$\begin{aligned} G_n^{1z}(s) &= 1 - G_{n-1}^{1z}(s) + \frac{G_{n-1}^{1z}(s)G_{n-1}^{1z}(s)}{G_{n-1}^{1z}(s) + \frac{s}{2(2n+1)}} \\ G_n^{11}(s) &= G_{n-1}^{11}(s) - \frac{G_{n-1}^{11}(s)G_{n-1}^{1z}(s)}{G_{n-1}^{1z}(s) + \frac{s}{2(2n+1)}} \\ G_n^{zz}(s) &= G_{n-1}^{zz}(s) - \frac{G_{n-1}^{zz}(s)G_{n-1}^{1z}(s)}{G_{n-1}^{1z}(s) + \frac{s}{2(2n+1)}} \\ G_n^{z1}(s) &= G_{n-1}^{z1}(s) - (-1)^{n-1} \frac{G_{n-1}^{11}(s)G_{n-1}^{zz}(s)}{G_{n-1}^{1z}(s) + \frac{s}{2(2n+1)}} \end{aligned}$$

We can analyze each term separately.

Case $G_n^{1z}(s)$.

This will be the most important term, as it determines the denominator of the expressions. Simplifying the recurrence slightly gives

$$\begin{aligned} G_n^{1z}(s) &= 1 - G_{n-1}^{1z}(s) + \frac{G_{n-1}^{1z}(s)G_{n-1}^{1z}(s)}{G_{n-1}^{1z}(s) + \frac{s}{2(2n+1)}} \\ &= \frac{(G_{n-1}^{1z}(s) + \frac{s}{2(2n+1)}) - \frac{s}{2(2n+1)} \cdot G_{n-1}^{1z}(s)}{G_{n-1}^{1z}(s) + \frac{s}{2(2n+1)}}. \end{aligned}$$

Now let $G_n^{1z}(s) = \frac{P_n^{1z}(s)}{Q_n^{1z}(s)}$ where P, Q are polynomials. Clearing the denominator Q yields

$$\frac{P_n^{1z}(s)}{Q_n^{1z}(s)} = \frac{(P_{n-1}^{1z}(s) + \frac{s}{2(2n+1)}Q_{n-1}^{1z}(s)) - \frac{s}{2(2n+1)} \cdot P_{n-1}^{1z}(s)}{P_{n-1}^{1z}(s) + \frac{s}{2(2n+1)} \cdot Q_{n-1}^{1z}(s)}.$$

This results in the recurrence

$$\begin{aligned} Q_n^{1z}(s) &= P_{n-1}^{1z}(s) + \frac{s}{2(2n+1)} \cdot Q_{n-1}^{1z}(s) \\ P_n^{1z}(s) &= Q_{n-1}^{1z}(s) - \frac{s}{2(2n+1)} \cdot P_{n-1}^{1z}(s). \end{aligned}$$

But this is exactly the *fundamental recurrence formula* for continuants of the continued fraction

$$e^s = 1 + \cfrac{s}{1 - \cfrac{\frac{1}{2}s}{1 + \cfrac{\frac{1}{6}s}{1 - \cfrac{\frac{1}{6}s}{1 + \cfrac{\frac{1}{10}s}{1 - \cfrac{\frac{1}{10}s}{1 + \ddots}}}}}}$$

Therefore $Q_{n-1}^{1z}(s)$ are the denominators of the Pade approximants.

Note that by definition of P, Q ,

$$G_{n-1}^{1z}(s) + \frac{s}{2(2n+1)} = \frac{P_{n-1}^{1z}(s) + \frac{s}{2(2n+1)} \cdot Q_{n-1}^{1z}(s)}{Q_{n-1}^{1z}(s)} = \frac{Q_n^{1z}(s)}{Q_{n-1}^{1z}(s)}$$

Going forward we will also suppress the superscript of Q , $Q_{n-1}(s) := Q_{n-1}^{1z}(s)$, as it will be evident that all terms have the same denominator $Q_n(s)$

Case $G_n^{11}(s)$.

First note that $G_n^{11}(s) = G_n^{zz}(s)$ is straightforward from the fact that their recurrences are

identical. The recurrence is

$$\begin{aligned}
 G_n^{11}(s) &= G_{n-1}^{11}(s) - \frac{G_{n-1}^{11}(s)G_{n-1}^{1z}(s)}{G_{n-1}^{1z}(s) + \frac{s}{2(2n+1)}} \\
 &= \frac{\frac{s}{2(2n+1)} \cdot G_{n-1}^{11}(s)}{G_{n-1}^{1z}(s) + \frac{s}{2(2n+1)}} \\
 &= \frac{\frac{s}{2(2n+1)} \cdot G_{n-1}^{11}(s)Q_{n-1}(s)}{P_{n-1}^{1z}(s) + \frac{s}{2(2n+1)} \cdot Q_{n-1}(s)} \\
 &= \frac{\frac{s}{2(2n+1)} \cdot G_{n-1}^{11}(s)Q_{n-1}(s)}{Q_n(s)}
 \end{aligned}$$

Therefore

$$\begin{aligned}
 G_n^{11}(s)Q_n(s) &= \frac{s}{2(2n+1)} \cdot G_{n-1}^{11}(s)Q_{n-1}(s) = \prod_{i=1}^n \frac{s}{2(2i+1)} \\
 G_n^{11}(s) &= \frac{\prod_{i=1}^n \frac{s}{2(2i+1)}}{Q_n(s)}
 \end{aligned}$$

Case $G_n^{z1}(s)$.

Define

$$G_n^{z1}(s) = \frac{P_n^{z1}(s)}{Q_n(s)}$$

This term satisfies the formula

$$\begin{aligned}
G_n^{z1}(s) &= G_{n-1}^{z1}(s) - (-1)^{n-1} \frac{G_{n-1}^{11}(s)G_{n-1}^{zz}(s)}{G_{n-1}^{1z}(s) + \frac{s}{2(2n+1)}} \\
&= \frac{P_{n-1}^{z1}(s)}{Q_{n-1}(s)} - (-1)^{n-1} \frac{\left(\prod_{i=0}^{n-1} \frac{s}{2(2i+1)}\right)^2 / Q_{n-1}(s)^2}{\frac{Q_n(s)}{Q_{n-1}(s)}} \\
&= \frac{P_{n-1}^{z1}(s)Q_n(s)}{Q_{n-1}(s)Q_n(s)} - (-1)^{n-1} \frac{\left(\prod_{i=0}^{n-1} \frac{s}{2(2i+1)}\right)^2}{Q_n(s)Q_{n-1}(s)}
\end{aligned}$$

By definition of P^{z1} ,

$$P_{n-1}^{z1}(s)Q_n(s) - (-1)^{n-1} \left(\prod_{i=0}^{n-1} \frac{s}{2(2i+1)} \right)^2 = P_n^{z1}(s)Q_{n-1}(s)$$

But note that this is exactly satisfied by the Padé approximants, by the determinantal formula of continued fractions. This shows that $G_{n-1}^{1z}(s)$ are the Padé approximants of e^{-s} , as desired.

E.5 Normalization and Timescales

Lemma E.15 (Closure properties of TO-SSMs). *Consider a TO-SSM (\mathbf{A}, \mathbf{B}) for basis functions $p_n(t)$ and measure $\omega(t)$. Then, the following are also TO-SSMs with the corresponding basis functions and measure:*

1. *Constant scaling changes the timescale: $(c\mathbf{A}, c\mathbf{B})$ is a TO-SSM with basis $p(ct)$ and measure $\omega(ct)c$.*
2. *Identity shift tilts by exponential: $(\mathbf{A} + c\mathbf{I}, \mathbf{B})$ is a TO-SSM with basis $p(t)e^{-ct}$ and measure $\omega(t)e^{2ct}$.*
3. *Unitary change of basis preserves measure: $(\mathbf{V}\mathbf{A}\mathbf{V}^*, \mathbf{V}\mathbf{B})$ is a TO-SSM with basis $\mathbf{V}p(t)$ and measure $\omega(t)$.*

Proof. We define $p(t)$ to be the vector of basis functions for the O-SSM (\mathbf{A}, \mathbf{B}) ,

$$p(t) = \begin{bmatrix} p_0(t) \\ \vdots \\ p_{N-1}(t) \end{bmatrix}.$$

Recall that the SSM kernels are $K_n(t) = p_n(t)\omega(t)$ so that $p(t)\omega(t) = e^{t\mathbf{A}}\mathbf{B}$.

1. The SSM kernels are

$$e^{t(c\mathbf{A})}(c\mathbf{B}) = ce^{(ct)\mathbf{A}}\mathbf{B} = cp(ct)\omega(ct).$$

It remains to show that the $p_n(ct)$ are orthonormal with respect to measure $c\omega(ct)$:

$$\int p_j(ct)p_k(ct)\omega(ct)c = \delta_{jk}$$

which follows immediately from the change of variables formula.

2. Using the commutativity of \mathbf{A} and \mathbf{I} , the SSM kernels are

$$e^{t(\mathbf{A}+c\mathbf{I})}\mathbf{B} = e^{t\mathbf{A}}e^{ct\mathbf{I}}\mathbf{B} = e^{ct}p(t)\omega(t).$$

It remains to show that $p_n(t)e^{-ct}$ are orthonormal with respect to measure $\omega(t)e^{2ct}$:

$$\int p_j(t)e^{-ct}p_k(t)e^{-ct}\omega(t)e^{2ct} = \int p_j(t)p_k(t)\omega(t) = \delta_{jk}.$$

3. The SSM basis is

$$e^{t\mathbf{V}\mathbf{A}\mathbf{V}^*}\mathbf{V}\mathbf{B} = \mathbf{V}e^{t\mathbf{A}}\mathbf{B} = \mathbf{V}p(t)\omega(t).$$

It remains to show that the basis functions $\mathbf{V}p(t)$ are orthonormal with respect to $\omega(t)$. Note that orthonormality of a set of basis functions can be expressed as $\int p(t)\omega(t)p(t)^\top = \mathbf{I}$,

so that

$$\begin{aligned}\int (\mathbf{V}p(t))\omega(t)(\mathbf{V}p(t))^* &= \mathbf{V} \left[\int p(t)\omega(t)p(t)^\top \right] \mathbf{V}^* \\ &= \mathbf{I}.\end{aligned}$$

□

Appendix F

Appendix for Chapter 6

F.1 Statements and Proofs for Alternative SSM Structures

This section proves the results in Section 6.1. These results are about efficiency results for alternative SSM structures outside of the main S4-Diag and S4-DPLR structures in the rest of this thesis.

- Appendix F.1.1 defines the class of low recurrence width (LRW) matrices, which is the class of matrices used to characterize the structure of general HIPPO operators with fast recurrent mode (Theorem 6.1).
- Theorem F.4 provides the formal statement of Theorem 6.1, which says that the hippo operators for any measure lead to a simple linear ODE of the form of equation (2.1a).
- Appendix F.1.3 proves Theorem 6.4, including a formal definition of quasiseparable matrices in Definition F.5. It also proves Corollary 6.5 showing that this algorithm is not numerically feasible to compute.

F.1.1 Recurrence Width

In Theorem 6.1, our final goal is to show that $x'(t) = \mathbf{A}(t)x(t) + \mathbf{B}(t)u(t)$ for some $\mathbf{A}(t)$ with constant recurrence width (see Definition F.1). Then in order to show that $\bar{\mathbf{A}}$ has MVM in $\tilde{O}(N)$ time, we use the fact that orthogonal polynomials all have recurrence width 2 and results regarding matrix-vector multiplication of matrices with constant recurrence width along with their inverses.

Definition F.1 ([41]). An $N \times N$ matrix \mathbf{A} has recurrence width t if the polynomials $a_i(X) = \sum_{j=0}^{N-1} \mathbf{A}[i,j]X^j$ satisfy $\deg(a_i) \leq i$ for $i < t$, and

$$a_i(X) = \sum_{j=1}^t g_{i,j}(X)a_{i-j}(X)$$

for $i \geq t$, where the polynomials $g_{i,j} \in \mathbb{R}[X]$ have degree at most j .

Theorem F.2 ([41], Theorem 4.4). For any $N \times N$ matrix \mathbf{A} with constant recurrence width, any vector $\mathbf{x} \in \mathbb{R}^n$, \mathbf{Ax} can be computed with $\tilde{O}(N)$ operations over \mathbb{R} .

Theorem F.3 ([41], Theorem 7.1). For any $N \times N$ matrix \mathbf{A} with constant recurrence width, any vector $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{A}^{-1}\mathbf{x}$ can be computed with $\tilde{O}(N)$ operations over \mathbb{R} .

For the rest of the note we'll assume that any operation over \mathbb{R} can be done in constant time. It would be useful for us to define $\mathbf{P} \in \mathbb{R}^{N \times N}$ such that the coefficients of the OP $p_i(X)$, i.e. $p_i(X) = \sum_{j=0}^{N-1} \mathbf{P}[i,j]X^j$.

F.1.2 General HIPPO Operators have Low Recurrence Width

With recurrence width defined, Theorem 6.1 is restated formally in Theorem F.4.

Theorem F.4. Let ω be any finitely-supported or one-sided measure with a corresponding orthogonal polynomial family (e.g., supported on $[-1, 1]$ or $[0, \infty)$ like the Jacobi or Laguerre classical orthogonal polynomial families). Let the HIPPO tilting be $\chi^{(t)}(Y) = \omega^{(t)}(Y)$ (from the general HIPPO framework, equation (D.2)).

Then the operator $\text{hippo}(\omega)$ is

$$\dot{x}_n(t) \approx -\frac{1}{\theta} \mathbf{Ax}(t) + \frac{2}{\theta} u(t) \begin{bmatrix} \vdots \\ p_n(1) \\ \vdots \end{bmatrix}$$

for some matrix \mathbf{A} that can be written as $\mathbf{A} = \mathbf{A}_1 \mathbf{A}_2 \mathbf{A}_3 + \mathbf{A}_4$, where \mathbf{A}_1 and \mathbf{A}_1 have recurrence width at most 2, \mathbf{A}_3 is the inverse of a matrix that has recurrence width 2, and \mathbf{A}_4 is rank-1.

F.1.3 Efficient State Space Kernel for Quasiseparable Matrices

We first define the quasiseparable matrices used in the statement of Theorem 6.4. The proof of Theorem 6.4 is omitted; see the original paper [74] for the full proof.

Definition F.5 (from [53]). *A matrix $\mathbf{R} \in \mathbb{R}^{N \times N}$ is (p, q) -quasiseparable if*

- *Every matrix contained strictly above the diagonal has rank at most p .*
- *Every matrix contained strictly below the diagonal has rank at most q .*

A (q, q) -quasiseparable matrix is called q -quasiseparable.

A nice property of (p, q) -quasiseparable matrices is that they are closed under inversion. Clearly, adding a diagonal matrix to a quasiseparable matrix also does not change quasiseparability, from the definition. This implies if \mathbf{A} is quasiseparable, then the bilinear-discretized matrix

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

is still quasiseparable.

We prove Theorem 6.4, by providing an algorithm to compute the state space kernel (2.8) efficiently for quasiseparable structured SSMs. We will show that

$$\mathcal{K}_L(\mathbf{A}, \mathbf{B}, \mathbf{C}) = (\mathbf{C}\mathbf{B}, \mathbf{C}\mathbf{A}\mathbf{B}, \dots, \mathbf{C}\mathbf{A}^{L-1}\mathbf{B}) \in \mathbb{R}^L$$

is efficiently computable for quasiseparable \mathbf{A} . By the above properties, this implies that the SSK $\mathcal{K}_L(\overline{\mathbf{A}}, \overline{\mathbf{B}}, \mathbf{C})$ is efficient when \mathbf{A} is quasiseparable.

The Algorithm

We follow the similar result of [41, Lemma 6.6] but track the dependence on L and the log factors more precisely, and optimize it in the case of stronger structure than quasiseparability, which holds in our setting.

The first step is to observe that the kernel function $\mathcal{K}_L(\mathbf{A}, \mathbf{B}, \mathbf{C})$ is actually the coefficient vector of $\mathbf{C}(\mathbf{I} - \mathbf{A}x)^{-1}\mathbf{B}$ (mod x^L) as a polynomial in x . (Note that $\mathbf{A}x$ means simply multiplying every entry in \mathbf{A} by a scalar variable x .) This follows from expanding the power series $(\mathbf{I} - \mathbf{A}x)^{-1} = \mathbf{I} + \mathbf{A}x + \mathbf{A}^2x^2 + \dots$. Thus we first compute $\mathbf{C}(\mathbf{I} - \mathbf{A}x)^{-1}\mathbf{B}$, which is

a rational function of degree at most N in the numerator and denominator (which can be seen by the standard adjoint formula for the matrix inverse).

The second step is simply inverting the denominator of this rational function $(\bmod x^L)$ and multiplying by the numerator, both of which are operations that need $L \log(L)$ time by standard results for polynomial arithmetic [189].

For the remainder of this section, we focus on computing the first part. We make two notational changes: First, we transpose \mathbf{C} to make it have the same shape as \mathbf{B} . We consider the more general setting where \mathbf{B} and \mathbf{C} have multiple columns; this can be viewed as handling a “batch” problem with several queries for \mathbf{B}, \mathbf{C} at the same time.

Lemma F.6. *Let \mathbf{A} be a q -quasiseparable matrix. Then*

$$\mathbf{C}^T(\mathbf{I} - \mathbf{A}x)^{-1}\mathbf{B} \quad \text{where } \mathbf{A} \in \mathbb{R}^{N \times N} \quad \mathbf{B}, \mathbf{C} \in \mathbb{R}^{N \times k}$$

is a $k \times k$ matrix of rational functions of degree at most N , which can be computed in $O(q^3 \log^4 N)$ operations.

The main idea is that quasiseparable matrices are recursively *self-similar*, in that the principal submatrices are also quasiseparable, which leads to a divide-and-conquer algorithm.

In particular, divide $\mathbf{A} = \begin{bmatrix} \mathbf{A}_{00} & \mathbf{A}_{01} \\ \mathbf{A}_{10} & \mathbf{A}_{11} \end{bmatrix}$ into quadrants. Then by Definition F.5, $\mathbf{A}_{00}, \mathbf{A}_{11}$ are both q -quasiseparable and $\mathbf{A}_{01}, \mathbf{A}_{10}$ are rank q . Therefore the strategy is to view $\mathbf{I} - \mathbf{A}x$ as a low-rank perturbation of smaller quasiseparable matrices and reduce the problem to a simpler one, by applying the Woodbury Matrix Identity (Proposition A.2). This identity holds for matrices over any ring \mathcal{R} ; unlikely a normal usage of this identity where \mathcal{R} is typically the base field like \mathbb{R} or \mathbb{C} , we will need \mathcal{R} to be the ring of *rational functions over \mathbb{R}* .

Proof of Lemma F.6. Since \mathbf{A} is q -quasiseparable, we can write $\mathbf{A}_{10} = \mathbf{U}_L \mathbf{V}_L^T$ and $\mathbf{A}_{01} = \mathbf{U}_U \mathbf{V}_U^T$ where $\mathbf{U}, \mathbf{V} \in \mathbb{F}^{N \times q}$. Notice that we can write $\mathbf{I} - \mathbf{A}x$ as

$$\mathbf{I} - \mathbf{A}x = \begin{bmatrix} \mathbf{I} - \mathbf{A}_{00}x & \mathbf{0} \\ \mathbf{0} & \mathbf{I} - \mathbf{A}_{11}x \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{U}_U \\ \mathbf{U}_L & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{V}_L & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_U \end{bmatrix}^T x.$$

Suppose we know the expansions of each of

$$\mathbf{M}_1 \in \mathcal{R}^{k \times k} = \mathbf{C}^T \begin{bmatrix} \mathbf{I} - \mathbf{A}_{00}x & \mathbf{0} \\ \mathbf{0} & \mathbf{I} - \mathbf{A}_{11}x \end{bmatrix}^{-1} \mathbf{B} \quad (\text{F.1})$$

$$\mathbf{M}_2 \in \mathcal{R}^{k \times 2q} = \mathbf{C}^T \begin{bmatrix} \mathbf{I} - \mathbf{A}_{00}x & \mathbf{0} \\ \mathbf{0} & \mathbf{I} - \mathbf{A}_{11}x \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{0} & \mathbf{U}_U \\ \mathbf{U}_L & \mathbf{0} \end{bmatrix} \quad (\text{F.2})$$

$$\mathbf{M}_3 \in \mathcal{R}^{2q \times 2q} = \begin{bmatrix} \mathbf{V}_L & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_U \end{bmatrix}^T \begin{bmatrix} \mathbf{I} - \mathbf{A}_{00}x & \mathbf{0} \\ \mathbf{0} & \mathbf{I} - \mathbf{A}_{11}x \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{0} & \mathbf{U}_U \\ \mathbf{U}_L & \mathbf{0} \end{bmatrix} \quad (\text{F.3})$$

$$\mathbf{M}_4 \in \mathcal{R}^{2q \times k} = \begin{bmatrix} \mathbf{V}_L & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_U \end{bmatrix}^T \begin{bmatrix} \mathbf{I} - \mathbf{A}_{00}x & \mathbf{0} \\ \mathbf{0} & \mathbf{I} - \mathbf{A}_{11}x \end{bmatrix}^{-1} \mathbf{B}. \quad (\text{F.4})$$

By Proposition A.2, the desired answer is

$$\mathbf{C}^T(x - \mathbf{A})^{-1} \mathbf{B} = \mathbf{M}_1 - \mathbf{M}_2(\mathbf{I}_{2q} + \mathbf{M}_3)^{-1} \mathbf{M}_4.$$

Then the final result can be computed by inverting $\mathbf{I}_{2t} + \mathbf{M}_3$ ($O(q^3 N \log(N))$ operations), multiplying by $\mathbf{M}_2, \mathbf{M}_4$ ($O((kq^2 + k^2q)N \log(N))$ operations), and subtracting from \mathbf{M}_1 ($O(k^2 N \log(N))$ operations). This is a total of $O((q^3 + kq^2 + k^2q)N \log(N))$ operations. Note that when $k = O(q \log N)$, this becomes $O(q^3 N \log^3 N)$; we will use this in the analysis shortly.

To compute $\mathbf{M}_1, \mathbf{M}_2, \mathbf{M}_3, \mathbf{M}_4$, it suffices to compute the following:

$$\begin{aligned} \mathbf{C}_1^T(\mathbf{I} - \mathbf{A}_{00}x)^{-1} \mathbf{B}_0 && \mathbf{C}_1^T(\mathbf{I} - \mathbf{A}_{11}x)^{-1} \mathbf{B}_1 \\ \mathbf{C}_0^T(\mathbf{I} - \mathbf{A}_{00}x)^{-1} \mathbf{U}_U && \mathbf{C}_1^T(\mathbf{I} - \mathbf{A}_{11}x)^{-1} \mathbf{U}_L \\ \mathbf{V}_L^T(\mathbf{I} - \mathbf{A}_{00}x)^{-1} \mathbf{U}_U && \mathbf{V}_U^T(\mathbf{I} - \mathbf{A}_{11}x)^{-1} \mathbf{U}_L \\ \mathbf{V}_L^T(\mathbf{I} - \mathbf{A}_{00}x)^{-1} \mathbf{B}_0 && \mathbf{V}_U^T(\mathbf{I} - \mathbf{A}_{11}x)^{-1} \mathbf{B}_1. \end{aligned} \quad (\text{F.5})$$

But to compute those, it suffices to compute the following $(k+t) \times (k+t)$ matrices:

$$\begin{aligned} \left[\mathbf{C}_0 \quad \mathbf{V}_L \right]^T (\mathbf{I} - \mathbf{A}_{00}x)^{-1} \begin{bmatrix} \mathbf{B}_0 & \mathbf{U}_U \end{bmatrix} \\ \left[\mathbf{C}_1 \quad \mathbf{V}_U \right]^T (\mathbf{I} - \mathbf{A}_{11}x)^{-1} \begin{bmatrix} \mathbf{B}_1 & \mathbf{U}_L \end{bmatrix} \end{aligned} \quad (\text{F.6})$$

Since \mathbf{A}_{00} and \mathbf{A}_{11} have the same form as \mathbf{A} , this is two recursive calls of half the size. Notice that the size of the other input (dimensions of \mathbf{B}, \mathbf{C}) is growing, but when the

initial input is $k = 1$, it never exceeds $1 + q \log N$ (since they increase by q every time we go down a level). Earlier, we noticed that when $k = O(q \log N)$, the reduction step has complexity $O(q^3 N \log^3(N))$ for any recursive call. The recursion adds an additional $\log N$ multiplicative factor on top of this. \square

Corollary F.7. *Suppose q is a small constant. Then the cost of Lemma F.6 is $O(N \log^2(N))$ operations.*

This follows from a small optimization in the algorithm. Notice that in the recursion (F.5) and (F.6), the \mathbf{U}, \mathbf{V} matrices do not have to be appended if they already exist in \mathbf{B}, \mathbf{C} . For intuition, this happens in the case when \mathbf{A} is tridiagonal, so that \mathbf{U}, \mathbf{V} have the structure $(1, 0, \dots, 0)$, or the case when the off-diagonal part of \mathbf{A} is all 1 (such as the HIPPO-LegT matrix). In fact, quasiseparable matrices do have this property in general, allowing one log factor to be removed.

Combining everything, this proves Theorem 6.4 with the exact bound $N \log^2(N) + L \log(L)$ operations. The memory claim follows similarly, and the depth of the algorithm is $\log^2(N) + \log(L)$ from the divide-and-conquer recursions. Finally, the space complexity is $O(N + L)$.

The Quasiseparable State Space Kernel is Numerically Unstable

However, as noted in Section 6.1, this algorithm is over exact arithmetic and unfortunately cannot be implemented in floating point. We now prove Corollary 6.5.

There are several reasons for the instability of this algorithm, but most directly we can pinpoint a particular intermediate quantity that involves exponentially large terms.

Corollary F.8. *The fast quasiseparable SSK algorithm computes coefficients of $p(x)$, the characteristic polynomial of \mathbf{A} , as an intermediate computation. Additionally, it computes the coefficients of its inverse, $p(x)^{-1} \pmod{x^L}$.*

We now claim that this quantity is numerically unfeasible. We narrow down to the case when $\bar{\mathbf{A}} = \mathbf{I}$ is the identity matrix. Note that this case is actually in some sense the most typical case: when discretizing the continuous-time SSM to discrete-time by a step-size Δ , the discretized transition matrix $\bar{\mathbf{A}}$ is brought closer to the identity. For example, with the Euler discretization $\bar{\mathbf{A}} = \mathbf{I} + \Delta \mathbf{A}$, we have $\bar{\mathbf{A}} \rightarrow \mathbf{I}$ as the step size $\Delta \rightarrow 0$.

Lemma F.9. *When $\overline{\mathbf{A}} = \mathbf{I}$, the fast quasiseparable SSK algorithm requires computing terms exponentially large in N .*

Proof. The characteristic polynomial of \mathbf{I} is

$$p(x) = \det |\mathbf{I} - x\mathbf{I}| = (1 - x)^N.$$

These coefficients have size up to $\binom{N}{\frac{N}{2}} \approx \frac{2^N}{\sqrt{\pi N/2}}$.

The inverse of $p(x)$ has even larger coefficients. It can be calculated in closed form by the generalized binomial formula:

$$(1 - x)^{-N} = \sum_{k=0}^{\infty} \binom{N + k - 1}{k} x^k.$$

Taking this $(\bmod x^L)$, the largest coefficient is

$$\binom{N + L - 2}{L - 1} = \binom{N + L - 2}{N - 1} = \frac{(L - 1)(L - 2) \dots (L - N + 1)}{(N - 1)!}.$$

When $L = N - 1$ this is

$$\binom{2(N - 1)}{N - 1} \approx \frac{2^{2N}}{\sqrt{\pi N}}$$

already larger than the coefficients of $(1 - x)^N$, and only increases as L grows. \square

F.2 Proofs for DPLR SSMs

This section proves the claims made in Section 6.2 about forms (diagonal and DPLR) of the HIPPO matrices.

F.2.1 HIPPO Diagonalization

We derive the explicit diagonalization of the HIPPO(-LegS) matrix, confirming the infeasibility of this diagonalization because of exponentially large entries.

Proof of Lemma 6.6. The HIPPO matrix (6.1) is equal, up to sign and conjugation by a

diagonal matrix, to

$$\mathbf{A} = \begin{bmatrix} 1 & & & & & & & \\ -1 & 2 & & & & & & \\ 1 & -3 & 3 & & & & & \\ -1 & 3 & -5 & 4 & & & & \\ 1 & -3 & 5 & -7 & 5 & & & \\ -1 & 3 & -5 & 7 & -9 & 6 & & \\ 1 & -3 & 5 & -7 & 9 & -11 & 7 & \\ -1 & 3 & -5 & 7 & -9 & 11 & -13 & 8 \\ \vdots & & & & & & & \ddots \end{bmatrix}$$

$$\mathbf{A}_{nk} = \begin{cases} (-1)^{n-k}(2k+1) & n > k \\ k+1 & n = k \\ 0 & n < k \end{cases}.$$

Our goal is to show that this \mathbf{A} is diagonalized by the matrix

$$\mathbf{V} = \binom{i+j}{i-j}_{ij} = \begin{bmatrix} 1 & & & & & & & \\ 1 & 1 & & & & & & \\ 1 & 3 & 1 & & & & & \\ 1 & 6 & 5 & 1 & & & & \\ 1 & 10 & 15 & 7 & 1 & & & \\ 1 & 15 & 35 & 28 & 9 & 1 & & \\ \vdots & & & & & & & \ddots \end{bmatrix},$$

or in other words that columns of this matrix are eigenvectors of \mathbf{A} .

Concretely, we will show that the j -th column of this matrix $\mathbf{v}^{(j)}$ with elements

$$\mathbf{v}_i^{(j)} = \begin{cases} 0 & i < j \\ \binom{i+j}{i-j} = \binom{i+j}{2j} & i \geq j \end{cases}$$

is an eigenvector with eigenvalue $j + 1$. In other words we must show that for all indices $k \in [N]$,

$$(\mathbf{A}\mathbf{v}^{(j)})_k = \sum_i \mathbf{A}_{ki} \mathbf{v}_i^{(j)} = (j+1)\mathbf{v}_k^{(j)}. \quad (\text{F.7})$$

If $k < j$, then for all i inside the sum, either $k < i$ or $i < j$. In the first case $\mathbf{A}_{ki} = 0$ and in the second case $\mathbf{v}_i^{(j)} = 0$, so both sides of equation (F.7) are equal to 0.

It remains to show the case $k \geq j$, which proceeds by induction on k . Expanding equation (F.7) using the formula for \mathbf{A} yields

$$(\mathbf{Av})_k^{(j)} = \sum_i \mathbf{A}_{ki} \mathbf{v}_i^{(j)} = \sum_{i=j}^{k-1} (-1)^{k-i} (2i+1) \binom{i+j}{2j} + (k+1) \binom{k+j}{2j}.$$

In the base case $k = j$, the sum disappears and we are left with $(\mathbf{Av}^{(j)})_j = (j+1) \binom{2j}{2j} = (j+1) \mathbf{v}_j^{(j)}$, as desired.

Otherwise, the sum for $(\mathbf{Av})_k^{(j)}$ is the same as the sum for $(\mathbf{Av})_{k-1}^{(j)}$ but with sign reversed and a few edge terms. The result follows from applying the inductive hypothesis and algebraic simplification:

$$\begin{aligned} (\mathbf{Av})_k^{(j)} &= -(\mathbf{Av})_{k-1}^{(j)} - (2k-1) \binom{k-1+j}{2j} + k \binom{k-1+j}{2j} + (k+1) \binom{k+j}{2j} \\ &= -(j+1) \binom{k-1+j}{2j} - (k-1) \binom{k-1+j}{2j} + (k+1) \binom{k+j}{2j} \\ &= -(j+k) \binom{k-1+j}{2j} + (k+1) \binom{k+j}{2j} \\ &= -(j+k) \frac{(k-1+j)!}{(k-1-j)!(2j)!} + (k+1) \binom{k+j}{2j} \\ &= -\frac{(k+j)!}{(k-1-j)!(2j)!} + (k+1) \binom{k+j}{2j} \\ &= -(k-j) \frac{(k+j)!}{(k-j)!(2j)!} + (k+1) \binom{k+j}{2j} \\ &= (j-k) \binom{k+j}{2j} + (k+1) \binom{k+j}{2j} \\ &= (j+1) \mathbf{v}_k^{(j)}. \end{aligned}$$

□

F.2.2 NPLR Representations of HIPPO Matrices

We prove Theorem 6.7, showing that all HIPPO matrices for continuous-time memory can be written in a normal plus low-rank (NPLR) representation.

We simultaneously show Corollary 6.8 and show that this NPLR form is Hurwitz.

Proof of Theorem 6.7 and Corollary 6.8. In order to show this result, it suffices to write each (real-valued) HIPPO matrix in the form $\mathbf{A} = \mathbf{N} - \mathbf{P}\mathbf{P}^\top$ where $\mathbf{N} \in \mathbb{R}^{N \times N}$ is a normal (real) matrix and $\mathbf{P} \in \mathbb{R}^{N \times r}$ for small r . Then $\mathbf{N} = \mathbf{V}\Lambda\mathbf{V}^*$ and \mathbf{A} would be unitarily similar to $\Lambda - (\mathbf{V}^*\mathbf{P})(\mathbf{V}^*\mathbf{P})^*$.

We consider each of the three cases HIPPO-LegT, HIPPO-LegS, and HIPPO-FouT separately.

HIPPO-LegS

We restate the formula from equation (5.1) for convenience.

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

Adding $\frac{1}{2}(2n+1)^{1/2}(2k+1)^{1/2}$ to the whole matrix gives

$$- \begin{cases} \frac{1}{2}(2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ \frac{1}{2} & \text{if } n = k \\ -\frac{1}{2}(2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n < k \end{cases}$$

Note that this matrix is not skew-symmetric, but is $\frac{1}{2}\mathbf{I} + \mathbf{S}$ where \mathbf{S} is a skew-symmetric matrix. This is diagonalizable by the same unitary matrix that diagonalizes \mathbf{S} .

It can be decomposed as

$$\begin{aligned} \mathbf{A} &= -\frac{1}{2}\mathbf{I} - \mathbf{S} - \mathbf{P}\mathbf{P}^* \\ \mathbf{S}_{nk} &= \begin{cases} \frac{1}{2}(2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ 0 & \text{if } n = k \\ -\frac{1}{2}(2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n < k \end{cases} \\ \mathbf{P}_n &= \left(n + \frac{1}{2}\right)^{1/2} \end{aligned}$$

Note that \mathbf{S} is skew-symmetric, and therefore normal. In turn $-\frac{1}{2}\mathbf{I} - \mathbf{S}$ is normal, providing the desired decomposition.

HIPPO-LegT

Up to a diagonal scaling by $\text{diag}(2n + 1)^{1/2}$, the LegT matrix (5.2) is

$$\mathbf{A} = -\begin{bmatrix} 1 & -1 & 1 & -1 & \dots \\ 1 & 1 & -1 & 1 & \\ 1 & 1 & 1 & -1 & \\ 1 & 1 & 1 & 1 & \\ \vdots & & & \ddots & \end{bmatrix} = -\begin{bmatrix} 0 & -1 & 0 & -1 & \dots \\ 1 & 0 & -1 & 0 & \\ 0 & 1 & 0 & -1 & \\ 1 & 0 & 1 & 0 & \\ \vdots & & & \ddots & \end{bmatrix} - \begin{bmatrix} 1 & 0 & 1 & 0 & \dots \\ 0 & 1 & 0 & 1 & \\ 1 & 0 & 1 & 0 & \\ 0 & 1 & 0 & 1 & \\ \vdots & & & \ddots & \end{bmatrix}.$$

The first term is skew-symmetric and the second term can be written as $\mathbf{P}\mathbf{P}^*$ for

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 1 & 0 & \dots \\ 0 & 1 & 0 & 1 & \dots \end{bmatrix}^\top$$

HIPPO-FouT

The FouT matrix (5.3) is

$$\begin{aligned} \mathbf{A} &= -\begin{bmatrix} 2 & 2\sqrt{2} & 0 & 2\sqrt{2} & 0 & \dots \\ 2\sqrt{2} & 4 & 2\pi & 4 & 0 & \dots \\ 0 & -2\pi & 0 & 0 & 0 & \dots \\ 2\sqrt{2} & 4 & 0 & 4 & 4\pi & \dots \\ 0 & 0 & 0 & -4\pi & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \\ &= -\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 2\pi & 0 & 0 & \dots \\ 0 & -2\pi & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 4\pi & \dots \\ 0 & 0 & 0 & -4\pi & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} - \begin{bmatrix} 2 & 2\sqrt{2} & 0 & 2\sqrt{2} & 0 & \dots \\ 2\sqrt{2} & 4 & 0 & 4 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots \\ 2\sqrt{2} & 4 & 0 & 4 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \end{aligned}$$

The first term is skew-symmetric and the second term can be written as $\mathbf{P}\mathbf{P}^*$ for

$$\mathbf{P} = \begin{bmatrix} \sqrt{2} & 2 & 0 & 2 & 0 & \dots \end{bmatrix}^\top$$

□

F.3 Proofs and Discussion of S4D Initializations

F.3.1 Proofs

We prove Theorem 6.9, and then show why this is a surprising result that is not true in general to low-rank perturbations of SSMs.

We start with the interpretation of the LegS matrices (\mathbf{A}, \mathbf{B}) shown in Chapter 5, which corresponds to Figure 3.1 (Left). We restate the core result about these matrices for convenience.

Theorem F.10. *Let $\mathbf{A}, \mathbf{B}, \mathbf{P}$ be the matrices defined in equation (6.3). The SSM kernels $K_n(t) = \mathbf{e}_n^\top e^{t\mathbf{A}} \mathbf{B}$ have the closed form formula*

$$K_n(t) = L_n(e^{-t})e^{-t}$$

where L_n are the Legendre polynomials shifted and scaled to be orthonormal on the interval $[0, 1]$.

Lemma F.11. *The functions $L_n(e^{-t})$ are a complete orthonormal basis with respect to the measure $\omega(t) = e^{-t}$.*

Proof. The polynomials are defined to be orthonormal on $[0, 1]$, i.e.

$$\int_0^1 L_n(t)L_m(t) dt = \delta_{n,m}.$$

By the change of variables $t = e^{-s}$ with $\frac{dt}{ds} = -e^{-s}$,

$$-\int_{-\infty}^0 L_n(e^{-s})L_m(e^{-s})e^{-s} ds = \delta_{n,m} = \int_0^\infty L_n(e^{-s})L_m(e^{-s})e^{-s} ds$$

which shows the orthonormality.

Completeness follows from the fact that polynomials are complete. \square

Proof of Theorem 6.9. We start with the standard interpretation of SSMs as convolutional systems. The SSM $x'(t) = \mathbf{A}x(t) + \mathbf{B}u(t)$ is equivalent to the convolution

$$x_n(t) = (u * K_n)(t) = \int_{-\infty}^t u(s)K_n(t-s) ds = \int_0^\infty u(t-s)K_n(s) ds$$

for the SSM kernels (equation (2.9)).

Defining $u^{(t)}(s) = u(t-s)$, we can write this as

$$x_n(t) = \langle u^{(t)}, K_n \rangle_\omega$$

where $\omega(s) = e^{-s}$ and $\langle p(s), q(s) \rangle_\omega = \int_0^\infty p(s)q(s)\omega(s) ds$ is the inner product in the Hilbert space of L2 functions with respect to measure ω .

By Theorem F.10, the K_n are a complete orthonormal basis in this Hilbert space. There $x_n(t)$ represents a decomposition of the function $u^{(t)}$ with respect to this basis, and can be recovered as a linear combination of these projections

$$u^{(t)} = \sum_{n=0}^{\infty} x_n(t)K_n.$$

Pointwise over the inner times s ,

$$u^{(t)}(s) = \sum_{n=0}^{\infty} x_n(t)K_n(s).$$

This implies that

$$\begin{aligned} u(t) &= u^{(t)}(0) = \sum_{n=0}^{\infty} x_n(t)K_n(0) \\ &= \sum_{n=0}^{\infty} x_n(t)L_n(0) = \sum_{n=0}^{\infty} x_n(t)(2n+1)^{\frac{1}{2}} \\ &= \mathbf{B}^\top x(t) \end{aligned}$$

Intuitively, due to the function reconstruction interpretation of HIPPO (Chapter 5), we can approximate $u(t)$ using knowledge in the current state $x(t)$. There in the limit $N \rightarrow \infty$,

the original SSM is equivalent to

$$\begin{aligned}
x'(t) &= \mathbf{A}x(t) + \mathbf{B}u(t) \\
&= \mathbf{A}x(t) + \frac{1}{2}\mathbf{B}u(t) + \frac{1}{2}\mathbf{B}u(t) \\
&= \mathbf{A}x(t) + \frac{1}{2}\mathbf{B}\mathbf{B}^\top x(t) + \frac{1}{2}\mathbf{B}u(t) \\
&= \mathbf{A}x(t) + \mathbf{P}\mathbf{P}^\top x(t) + \frac{1}{2}\mathbf{B}u(t) \\
&= \mathbf{A}^N x(t) + \frac{1}{2}\mathbf{B}u(t)
\end{aligned}$$

□

F.3.2 General Low-rank Perturbations

Finally, we remark that the phenomenon described by Theorem 6.9, where removing the low-rank correction to a DPLR matrix approximates the original dynamics, is unique to this HIPPO-LegS matrix. We note that if instead of $\mathbf{P}\mathbf{P}^\top$, a *random* rank-1 correction is added to the HIPPO-LegS matrix in Theorem 6.9, the resulting SSM kernels look completely different and in fact diverge rapidly as the magnitude of \mathbf{P} increases (Figure F.1).

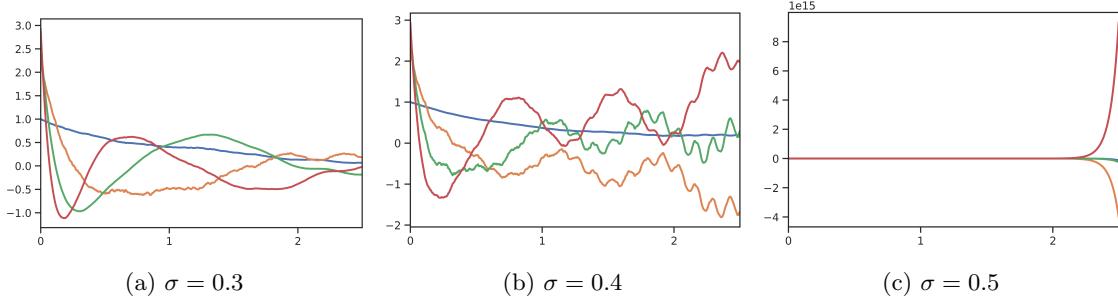


Figure F.1: Basis kernels for $(\mathbf{A} + \mathbf{P}\mathbf{P}^\top, \mathbf{B})$ for HIPPO-LegS (\mathbf{A}, \mathbf{B}) and random i.i.d. Gaussian \mathbf{P} with varying std σ , illustrating that the SSM basis is very sensitive to low-rank perturbations. Note that the normal-HIPPO matrix $\mathbf{A}^{(N)} = \mathbf{A} + \mathbf{P}\mathbf{P}^\top$ for \mathbf{P} with entries of magnitude $N^{\frac{1}{2}}$ which is far larger, highlighting how unexpected the theoretical result Theorem 6.9 is.

Similarly, Figure F.2a shows S4-FouT which is also DPLR, but removing the low-rank component dramatically changes the SSM kernels.

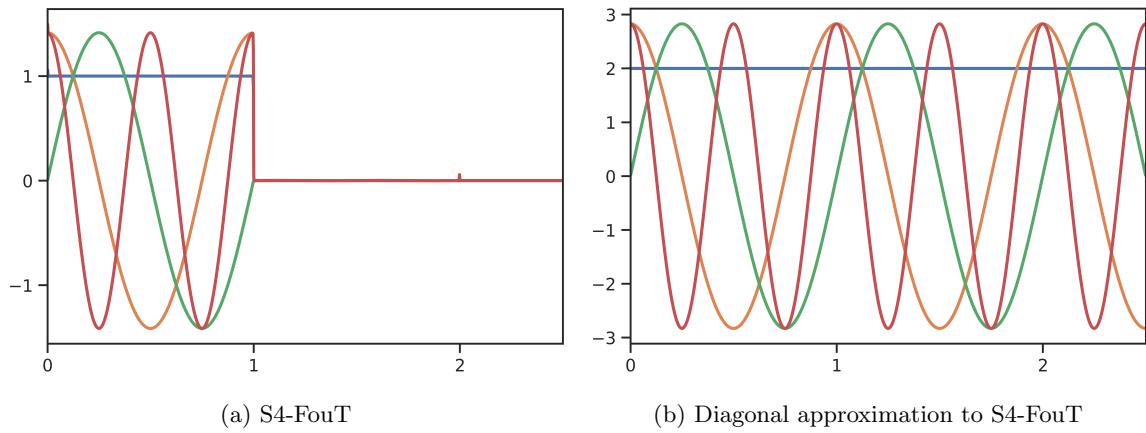


Figure F.2: (a) S4-FouT is an instantiation of S4-DPLR choosing a particular (\mathbf{A}, \mathbf{B}) that produces truncated Fourier basis functions. This captures sliding Fourier transforms as a state space model. (b) Removing the low-rank term from the FouT matrix does *not* approximate S4-FouT. This diagonal state matrix has real part 0 that produces infinite oscillations and does not perform well empirically.

Appendix G

Appendix for Chapter 7

This chapter contains experimental details and protocols, hyperparameters, and extended results and citations for our experimental evaluation in Chapter 7.

Appendix G.1 and Appendix G.2 contain details for Section 7.3 and Section 7.4 respectively.

G.1 General Sequence Modeling

This section corresponds to the experiments in Section 7.3.

- Appendix G.1.1 discusses general training protocol and guidelines, particularly for the main classification and regression tasks.
- Appendix G.1.2 contains more specific training details for the classification and regression tasks in Section 7.3.1, Section 7.3.2, Section 7.3.3, and Section 7.3.4.
- Appendix G.1.3 includes details of baselines that we trained, particularly for Section 7.3.3.
- Appendix G.1.4 has additional results for Section 7.3.5.
- Appendix G.1.5 has training details for CIFAR-10 density estimation in Section 7.3.6.
- Appendix G.1.6 has training details for WikiText-103 language modeling in Section 7.3.6.

G.1.1 General Training Protocol

We first discuss our general training methodology applied to almost all tasks. The next sections will discuss details and exceptions for specific tasks.

Optimizer

All models are trained with the AdamW optimizer [109, 133] with a learning rate schedule (almost always cosine decay, except for exceptions noted below).

Regularization

We generally use no dropout and mild weight decay for regularization (a good default used for most tasks is 0.05 weight decay).

A variant of dropout that ties the mask across the sequence length [206] can help; it is only used on the sequential image tasks.

Classification and Regression

Classification tasks use the cross-entropy loss and regression tasks use mean squared error (Section 2.1). Since the output of our sequence models (which map sequences to sequences) have a length dimension in the output, and basic classification/regression tasks have a single target per example, mean pooling along the length dimension is applied after the last layer.

Since these tasks do not require causality, the bidirectional version of the S4 layer is used (Section 7.2.6).

Architecture

Unless otherwise specified, we use an SSNN composed of repeating the architecture block described in Section 7.2. The main axes of variation include the width (H) and depth, the normalization placement (pre-norm vs. post-norm), and the normalization type (BatchNorm vs. LayerNorm).

S4 Layer

Within an S4 layer, the state size is almost always set to $N = 64$.

Table G.1: Hyperparameters for Tables 7.1 and 7.3a. LR is learning rate and WD is weight decay. BN and LN refer to Batch Normalization and Layer Normalization.

	Depth	Features H	Norm	Pre-norm	Dropout	LR	Batch Size	Epochs	WD	Patience
CIFAR-10	6	1024	LN	False	0.25	0.01	50	200	0.01	20
Speech Commands (MFCC)	4	256	LN	False	0.2	0.01	100	50	0	5
Speech Commands (Raw)	6	128	BN	True	0.1	0.01	20	150	0	10

The learning rate on parameters governing the ODE or recurrence part of the equation (2.1a) (in particular Λ, P, B, Δ) are reduced to a maximum starting LR of 0.001, which improves stability since the dynamics are most sensitive to the A (and \bar{A}) parameter.

Hardware

With the exception of the larger scale experiments in Section 7.3.6, all models were run on single GPU. Some tasks used an A100 GPU (notably, the Path-X experiments), which has a larger max memory of 40Gb. To reproduce these on smaller GPUs, the batch size can be reduced or gradients can be accumulated for two batches.

G.1.2 Classification and Regression

The experiments in Sections 7.3.1 to 7.3.4 are all classification and regression tasks that use the general default settings.

Image Classification and SC-10

Table 7.1 and Table 7.3a were results obtained with the first published version of S4, which had a slightly different training protocol. The main differences were using a constant learning rate scheduler that decayed on validation plateau (by a factor of 0.1 or 0.2), and using dropout instead of weight decay.

These hyperparameters are reported in Table G.1.

SC, BIDMC, Long Range Arena

The rest of the classification and speech tasks (Tables 7.2, 7.3b and 7.4) follow the general protocol with as few hyperparameters changed as possible.

Scheduler: The learning rate scheduler is a cosine annealing learning rate scheduler for all tasks, with linear warmup always set to 1/10 of the total training steps.

Regularization: Almost all tasks used no dropout and 0.05 weight decay.

Table G.2: The values of the best hyperparameters found for all datasets; full models on ablation datasets (Top) and LRA (Bottom). LR is learning rate and WD is weight decay. BN and LN refer to Batch Normalization and Layer Normalization.

	Depth	Features H	State Size N	Norm	Pre-norm	Dropout	LR	Batch Size	Epochs	WD	$(\Delta_{min}, \Delta_{max})$
BIDMC	6	128	256	LN	True	0	0.01	32	500	0.05	(0.001, 0.1)
SC	6	128	64	BN	True	0	0.01	16	40	0.05	(0.001, 0.1)
ListOps	6	256	4	BN	False	0	0.01	32	40	0.05	(0.001, 0.1)
Text	6	256	4	BN	True	0	0.01	16	32	0.05	(0.001, 0.1)
Retrieval	6	256	4	BN	True	0	0.01	64	20	0.05	(0.001, 0.1)
Image	6	512	64	BN	False	0.1	0.01	50	200	0.05	(0.001, 0.1)
Pathfinder	6	256	64	BN	True	0	0.004	64	200	0.05	(0.001, 0.1)
Path-X	6	256	64	BN	True	0	0.001	16	50	0.05	(0.0001, 0.1)

Architecture: Almost all tasks used an architecture with 6 layers, $H = 256$ hidden units, BatchNorm, and pre-norm placement of the normalization layer.

Exceptions to the above rules are described below. Full hyperparameters are in Table G.2.

LRA Image. We used a larger number of hidden features H , post-norm instead of pre-norm, and minor dropout. We note that the choice of normalization and increased H do not make a significant difference on final performance, still attaining classification accuracy in the high 80’s. Dropout does seem to make a difference.

BIDMC. We used a larger state size of $N = 256$, since we hypothesized that picking up higher frequency features on this dataset would help. We also used a step scheduler that decayed the LR by 0.5 every 100 epochs, following prior work [178].

ListOps. We also found that post-norm generalized better than pre-norm, but results are again close (less than 1% difference).

Path-X. The initialization range for Path-X is decreased from $(\Delta_{min}, \Delta_{max}) = (0.001, 0.1)$ to $(\Delta_{min}, \Delta_{max}) = (0.0001, 0.1)$. This is justified by the theory in Section 5.4, and was ablated on the Path-X dataset in Section 7.5.2.

Textual data. For the three LRA datasets with text data (ListOps, Text, Retrieval), we decrease the state size to $N = 4$, following similar hyperparameter choices from the related S5 [192] and Liquid-S4 [79] works.

Finally, we note that final accuracies do not necessarily convey all the performance characteristics and tradeoffs of methods. For example, earlier versions of the LRA experiments used a slightly different learning rate scheduler where the warmup was fixed to 1000 steps

instead of one-tenth of the total training length. With these settings, we initially found that S4D-Lin (as well as S4-Rand and S4D-Rand) did not learn at all on Path-X, while the initializations based on HIPPO (S4D-LegS and S4D-Inv) did. Many questions remain toward understanding the theoretical and empirical properties of these models.

G.1.3 Speech Commands Baselines

Table 7.3 contains numerous baselines that we trained from scratch. Details are reported in this section.

SC10 Baselines

We provide details of sweeps run for baseline methods run by us—numbers for all others method are taken from Gu et al. [74]. The best hyperparameters used for S4 are included in Table G.1.

Transformer [217] For MFCC, we swept the number of model layers {2, 4}, dropout {0, 0.1} and learning rates {0.001, 0.0005}. We used 8 attention heads, model dimension 128, prenorm, positional encodings, and trained for 150 epochs with a batch size of 100. For Raw, the Transformer model’s memory usage made training impossible.

Performer [30] For MFCC, we swept the number of model layers {2, 4}, dropout {0, 0.1} and learning rates {0.001, 0.0005}. We used 8 attention heads, model dimension 128, prenorm, positional encodings, and trained for 150 epochs with a batch size of 100. For Raw, we used a model dimension of 128, 4 attention heads, prenorm, and a batch size of 16. We reduced the number of model layers to 4, so the model would fit on the single GPU. We trained for 100 epochs with a learning rate of 0.001 and no dropout.

ExpRNN [121] For MFCC, we swept hidden sizes {256, 512} and learning rates

$$\{0.001, 0.002, 0.0005\}.$$

Training was run for 200 epochs, with a single layer model using a batch size of 100. For Raw, we swept hidden sizes {32, 64} and learning rates {0.001, 0.0005} (however, ExpRNN failed to learn).

LipschitzRNN [56] For MFCC, we swept hidden sizes {256, 512} and learning rates

$$\{0.001, 0.002, 0.0005\}.$$

Training was run for 150 epochs, with a single layer model using a batch size of 100. For Raw, we found that LipschitzRNN was too slow to train on a single GPU (requiring a full day for 1 epoch of training alone).

WaveGAN Discriminator [50] The WaveGAN-D in Table 7.3a is actually our improved version of the discriminator network from the recent WaveGAN model for speech [50]. One characteristic is that instead of mean pooling to reduce the output sequence to a single classification target, it used a fully-connected linear layer, and thus could not handle sequences of any other length. This architecture was updated to the ConvNet architecture used on the full SC dataset, described below.

SC Baseline: ConvNet Architecture

Our ConvNet architecture is a custom CNN created as a strong baseline for Speech Commands classification, comprising:

- Four stages, each composed of three identical residual blocks.
- The first stage has model dimension (i.e. channels, in CNN nomenclature) $H = 64$. Each stage doubles the dimension of the previous stage (with a position-wise linear layer) and ends in an average pooling layer of width 4. Thus, the first stage operates on inputs of length 16384, dimension 64 (the input is zero-padded from 16000 to 16384) and the last on length 256, dimension 512.
- Each residual block has a (pre-norm) BatchNorm layer followed by a convolution layer and GeLU activation.
- Convolution layers have a kernel size of 25.

G.1.4 Time Series Forecasting

Tables G.3 and G.4 contain full results on all 50 settings reported by Zhou et al. [241]. S4 sets the best results on 40 out of 50 of these settings.

G.1.5 CIFAR Density Estimation

This task used a different backbone than the rest of our experiments. We used blocks of alternating S4 layers and position-wise feed-forward layers (in the style of Transformer blocks). Each feed-forward intermediate dimension was set to $2 \times$ the hidden size of the incoming S4 layer. Similar to Salimans et al. [182], we used a UNet-style backbone consisting

Table G.3: Univariate long sequence time-series forecasting results on four datasets (five cases).

Methods	S4		Informer		Informer [†]		LogTrans		Reformer		LSTMa		DeepAR		ARIMA		Prophet	
Metric	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	24	0.061 0.191	0.098	0.247	0.092	0.246	0.103	0.259	0.222	0.389	0.114	0.272	0.107	0.280	0.108	0.284	0.115	0.275
	48	0.079 0.220	0.158	0.319	0.161	0.322	0.167	0.328	0.284	0.445	0.193	0.358	0.162	0.327	0.175	0.424	0.168	0.330
	168	0.104 0.258	0.183	0.346	0.187	0.355	0.207	0.375	1.522	1.191	0.236	0.392	0.239	0.422	0.396	0.504	1.224	0.763
	336	0.080 0.229	0.222	0.387	0.215	0.369	0.230	0.399	1.860	1.124	0.590	0.698	0.445	0.552	0.468	0.593	1.549	1.820
	720	0.116 0.271	0.269	0.435	0.257	0.421	0.273	0.463	2.112	1.436	0.683	0.768	0.658	0.707	0.659	0.766	2.735	3.253
ETTh2	24	0.095	0.234	0.093 0.240	0.099	0.241	0.102	0.255	0.263	0.437	0.155	0.307	0.098	0.263	3.554	0.445	0.199	0.381
	48	0.191	0.346	0.155 0.314	0.159	0.317	0.169	0.348	0.458	0.545	0.190	0.348	0.163	0.341	3.190	0.474	0.304	0.462
	168	0.167 0.333	0.232	0.389	0.235	0.390	0.246	0.422	1.029	0.879	0.385	0.514	0.255	0.414	2.800	0.595	2.145	1.068
	336	0.189 0.361	0.263	0.417	0.258	0.423	0.267	0.437	1.668	1.228	0.558	0.606	0.604	0.607	2.753	0.738	2.096	2.543
	720	0.187 0.358	0.277	0.431	0.285	0.442	0.303	0.493	2.030	1.721	0.640	0.681	0.429	0.580	2.878	1.044	3.355	4.664
ETTm1	24	0.024 0.117	0.030	0.137	0.034	0.160	0.065	0.202	0.095	0.228	0.121	0.233	0.091	0.243	0.090	0.206	0.120	0.290
	48	0.051 0.174	0.069	0.203	0.066	0.194	0.078	0.220	0.249	0.390	0.305	0.411	0.219	0.362	0.179	0.306	0.133	0.305
	96	0.086 0.229	0.194	0.372	0.187	0.384	0.199	0.386	0.920	0.767	0.287	0.420	0.364	0.496	0.272	0.399	0.194	0.396
	288	0.160 0.327	0.401	0.554	0.409	0.548	0.411	0.572	1.108	1.245	0.524	0.584	0.948	0.795	0.462	0.558	0.452	0.574
	672	0.292 0.466	0.512	0.644	0.519	0.665	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	24	0.125	0.254	0.117 0.251	0.119	0.256	0.136	0.279	0.231	0.401	0.131	0.254	0.128	0.274	0.219	0.355	0.302	0.433
	48	0.181	0.305	0.178 0.318	0.185	0.316	0.206	0.356	0.328	0.423	0.190	0.334	0.203	0.353	0.273	0.409	0.445	0.536
	168	0.198 0.333	0.266	0.398	0.269	0.404	0.309	0.439	0.654	0.634	0.341	0.448	0.293	0.451	0.503	0.599	2.441	1.142
	336	0.300	0.417	0.297 0.416	0.310	0.422	0.359	0.484	1.792	1.093	0.456	0.554	0.585	0.644	0.728	0.730	1.987	2.468
	720	0.245 0.375	0.359	0.466	0.361	0.471	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	48	0.222	0.350	0.239	0.359	0.238	0.368	0.280	0.429	0.971	0.884	0.493	0.539	0.204 0.357	0.879	0.764	0.524	0.595
	168	0.331	0.421	0.447	0.503	0.442	0.514	0.454	0.529	1.671	1.587	0.723	0.655	0.315 0.436	1.032	0.833	2.725	1.273
	336	0.328 0.422	0.489	0.528	0.501	0.552	0.514	0.563	3.528	2.196	1.212	0.898	0.414	0.519	1.136	0.876	2.246	3.077
	720	0.428 0.494	0.540	0.571	0.543	0.578	0.558	0.609	4.891	4.047	1.511	0.966	0.563	0.595	1.251	0.933	4.243	1.415
	960	0.432 0.497	0.582	0.608	0.594	0.638	0.624	0.645	7.019	5.105	1.545	1.006	0.657	0.683	1.370	0.982	6.901	4.264
Count	22	5	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	

of B identical blocks followed by a downsampling layer. The downsampling rates were 3, 4, 4 (the 3 chosen because the sequence consists of RGB pixels). The base model had $B = 8$ with starting hidden dimension 128, while the large model had $B = 16$ with starting hidden dimension 192.

We experimented with both the mixture of logistics from [182] as well as a simpler 256-way categorical loss. We found they were pretty close and ended up using the simpler softmax loss along with using input embeddings.

We used the LAMB optimizer with learning rate 0.005. The base model had no dropout, while the large model had dropout 0.1 before the linear layers inside the S4 and FF blocks.

G.1.6 WikiText-103 Language Modeling

The RNN baselines included in Table 7.7 are the AWD-QRNN [137], an efficient linear gated RNN, and the LSTM + Cache + Hebbian + MbPA [169], the best performing pure RNN in the literature. The CNN baselines are the CNN with GLU activations [38], the TrellisNet [12], Dynamic Convolutions [227], and TaLK Convolutions [126].

The Transformer baseline is [8], which uses Adaptive Inputs with a tied Adaptive Softmax. This model is a standard high-performing Transformer baseline on this benchmark, used

Table G.4: Multivariate long sequence time-series forecasting results on four datasets (five cases).

Methods	S4				Informer		Informer [†]		LogTrans		Reformer		LSTMa		LSTnet	
Metric	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	24	0.525	0.542	0.577	0.549	0.620	0.577	0.686	0.604	0.991	0.754	0.650	0.624	1.293	0.901	
	48	0.641	0.615	0.685	0.625	0.692	0.671	0.766	0.757	1.313	0.906	0.702	0.675	1.456	0.960	
	168	0.980	0.779	0.931	0.752	0.947	0.797	1.002	0.846	1.824	1.138	1.212	0.867	1.997	1.214	
	336	1.407	0.910	1.128	0.873	1.094	0.813	1.362	0.952	2.117	1.280	1.424	0.994	2.655	1.369	
	720	1.162	0.842	1.215	0.896	1.241	0.917	1.397	1.291	2.415	1.520	1.960	1.322	2.143	1.380	
ETTh2	24	0.871	0.736	0.720	0.665	0.753	0.727	0.828	0.750	1.531	1.613	1.143	0.813	2.742	1.457	
	48	1.240	0.867	1.457	1.001	1.461	1.077	1.806	1.034	1.871	1.735	1.671	1.221	3.567	1.687	
	168	2.580	1.255	3.489	1.515	3.485	1.612	4.070	1.681	4.660	1.846	4.117	1.674	3.242	2.513	
	336	1.980	1.128	2.723	1.340	2.626	1.285	3.875	1.763	4.028	1.688	3.434	1.549	2.544	2.591	
	720	2.650	1.340	3.467	1.473	3.548	1.495	3.913	1.552	5.381	2.015	3.963	1.788	4.625	3.709	
ETTm1	24	0.426	0.487	0.323	0.369	0.306	0.371	0.419	0.412	0.724	0.607	0.621	0.629	1.968	1.170	
	48	0.580	0.565	0.494	0.503	0.465	0.470	0.507	0.583	1.098	0.777	1.392	0.939	1.999	1.215	
	96	0.699	0.649	0.678	0.614	0.681	0.612	0.768	0.792	1.433	0.945	1.339	0.913	2.762	1.542	
	288	0.824	0.674	1.056	0.786	1.162	0.879	1.462	1.320	1.820	1.094	1.740	1.124	1.257	2.076	
	672	0.846	0.709	1.192	0.926	1.231	1.103	1.669	1.461	2.187	1.232	2.736	1.555	1.917	2.941	
Weather	24	0.334	0.385	0.335	0.381	0.349	0.397	0.435	0.477	0.655	0.583	0.546	0.570	0.615	0.545	
	48	0.406	0.444	0.395	0.459	0.386	0.433	0.426	0.495	0.729	0.666	0.829	0.677	0.660	0.589	
	168	0.525	0.527	0.608	0.567	0.613	0.582	0.727	0.671	1.318	0.855	1.038	0.835	0.748	0.647	
	336	0.531	0.539	0.702	0.620	0.707	0.634	0.754	0.670	1.930	1.167	1.657	1.059	0.782	0.683	
	720	0.578	0.578	0.831	0.731	0.834	0.741	0.885	0.773	2.726	1.575	1.536	1.109	0.851	0.757	
ECL	48	0.255	0.352	0.344	0.393	0.334	0.399	0.355	0.418	1.404	0.999	0.486	0.572	0.369	0.445	
	168	0.283	0.373	0.368	0.424	0.353	0.420	0.368	0.432	1.515	1.069	0.574	0.602	0.394	0.476	
	336	0.292	0.382	0.381	0.431	0.381	0.439	0.373	0.439	1.601	1.104	0.886	0.795	0.419	0.477	
	720	0.289	0.377	0.406	0.443	0.391	0.438	0.409	0.454	2.009	1.170	1.676	1.095	0.556	0.565	
	960	0.299	0.387	0.460	0.548	0.492	0.550	0.477	0.589	2.141	1.387	1.591	1.128	0.605	0.599	
Count	18				5		6		0		0		0		0	

for example by Lioutas and Guo [126] and many more.

Our S4 model uses the same Transformer backbone as in [8]. The model consists of 16 blocks of S4 layers alternated with position-wise feedforward layers, with a feature dimension of 1024. Because our S4 layer has around 1/4 the number of parameters as a self-attention layer with the same dimension, we made two modifications to match the parameter count better: (i) we used a GLU activation after the S4 linear layer (Section 7.2) (ii) we used two S4 layers per block. Blocks use Layer Normalization in the pre-norm position. The embedding and softmax layers were the Adaptive Embedding from [8] with standard cutoffs 20000, 40000, 200000.

Evaluation was performed similarly to the basic setting in [8], Table 5, which uses sliding non-overlapping windows. Other settings are reported in [8] that include more context at training and evaluation time and improves the score. Because such evaluation protocols are orthogonal to the basic model, we do not consider them and report the base score from [8] Table 5.

Instead of SGD+Momentum with multiple cosine learning rate annealing cycles, our S4 model was trained with the simpler AdamW optimizer with a single cosine learning rate cycle with a maximum of 800000 steps. The initial learning rate was set to 0.0005. We used

8 A100 GPUs with a batch size of 1 per GPU and context size 8192. We used no gradient clipping and a weight decay of 0.1. Unlike [8] which specified different dropout rates for different parameters, we used a constant dropout rate of 0.25 throughout the network, including before every linear layer and on the residual branches.

G.2 Additional Capabilities of S4

Appendix G.2.1 contains details for the autoregressive generation speed results in Section 7.4.2 Appendix G.2.2 has details for the benchmarking results in Section 7.4.3.

G.2.1 Efficient Inference: Autoregressive Generation Details

Protocol

To account for different model sizes and memory requirements for each method, we benchmark generation speed by throughput, measured in images per second (Table 7.6) or tokens per second (Table 7.7). Each model generates images on a single A100 GPU, maximizing batch size to fit in memory. (For CIFAR-10 generation we limited memory to 16Gb, to be more comparable to the Transformer and Linear Transformer results reported from [104].)

Baselines

The Transformer and Linear Transformer baselines reported in Table 7.6 are the results reported directly from Katharopoulos et al. [104]. Note that the Transformer number is the one in their Appendix, which implements the optimized cached implementation of self-attention.

For all other baseline models, we used open source implementations of the models to benchmark generation speed. For the PixelCNN++, we used the fast cached version by Ramachandran et al. [170], which sped up generation by orders of magnitude from the naive implementation. This code was only available in TensorFlow, which may have slight differences compared to the rest of the baselines which were implemented in PyTorch.

We were unable to run the Sparse Transformer [27] model due to issues with their custom CUDA implementation of the sparse attention kernel, which we were unable to resolve.

The Transformer baseline from Table 7.7 was run using a modified GPT-2 backbone from the HuggingFace repository, configured to recreate the architecture reported in [8]. These

numbers are actually slightly favorable to the baseline, as we did not include the timing of the embedding or softmax layers, whereas the number reported for S4 is the full model.

G.2.2 Efficient Training: Benchmarking Details

Benchmarking results from Table 7.8 and Table 7.9 were tested on a single A100 GPU.

Benchmarks against Naive SSM

For a given dimension H , a single LSSL or S4 layer was constructed with H hidden features. For LSSL, the state size N was set to H as done in [74]. For S4, the state size N was set to parameter-match the LSSL, which was a state size of $\frac{N}{4}$ due to differences in the parameterization. Table 7.8 benchmarks a single forward+backward pass of a single layer.

Benchmarks against Efficient Transformers

Following [202], the Transformer models had 4 layers, hidden dimension 256 with 4 heads, query/key/value projection dimension 128, and batch size 32, for a total of roughly $600k$ parameters. The S4 model was parameter matched while keeping the depth and hidden dimension constant (leading to a state size of $N = 256$).

We note that the relative orderings of these methods can vary depending on the exact hyperparameter settings.

Appendix H

Appendix for Chapter 8

H.1 Additional Results

We provide details of ablations, including architecture ablations and efficiency benchmarking.

YouTubeMix

We conduct architectural ablations and efficiency benchmarking for all baselines on the YouTubeMix dataset.

Architectures. SampleRNN-2 and SampleRNN-3 correspond to the 2- and 3-tier models described in Appendix H.2.2 respectively. WaveNet-512 and WaveNet-1024 refer to models with 512 and 1024 skip channels respectively with all other details fixed as described in Appendix H.2.2. SAShIMI-{2, 4, 6, 8} consist of the indicated number of S4 blocks in each tier of the architecture, with all other details being the same.

Isotropic S4. We also include an isotropic S4 model to ablate the effect of pooling in SAShIMI. Isotropic S4 can be viewed as SAShIMI without any pooling (i.e. no additional tiers aside from the top tier). We note that due to larger memory usage for these models, we use a sequence length of 4s for the 4 layer isotropic model, and a sequence length of 2s for the 8 layer isotropic model (both with batch size 1), highlighting an additional disadvantage in memory efficiency.

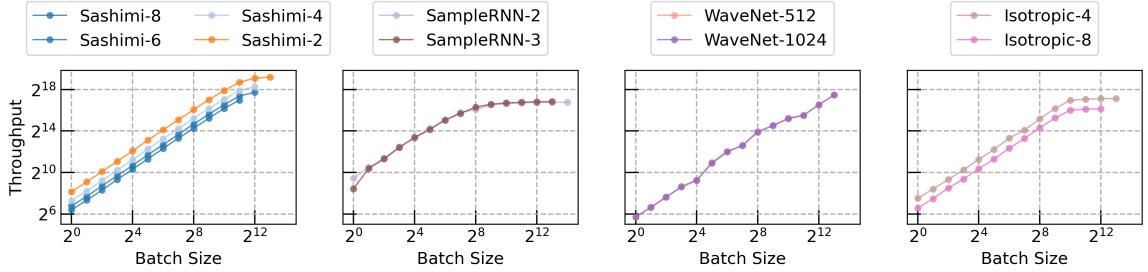


Figure H.1: Log-log plot of throughput vs. batch size. Throughput scales near linearly for SASHIMI. By contrast, SampleRNN throughput peaks at smaller batch sizes, while WaveNet shows sublinear scaling with throughput degradation at some batch sizes. Isotropic variants have far lower throughput than SASHIMI.

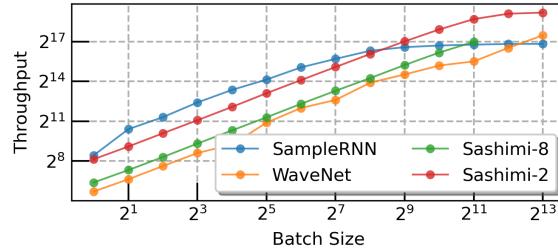


Figure H.2: Log-log plot of throughput vs. batch size. SASHIMI-2 improves peak throughput over WaveNet and SampleRNN by 3× and 5× respectively.

Throughput Benchmarking. To measure peak throughput, we track the time taken by models to generate 1000 samples at batch sizes that vary from 1 to 8192 in powers of 2. The throughput is the total number of samples generated by a model in 1 second. Figure H.1 shows the results of this study in more detail for each method.

Diffusion model ablations. Table 8.7 reports results for the ablations described in Section 8.4.3. Experimental details are provided in Appendix H.2.2.

H.2 Experiment Details

We include experimental details, including dataset preparation, hyperparameters for all methods, details of ablations as well as descriptions of automated and human evaluation metrics below.

H.2.1 Datasets

A summary of dataset information can be found in Table 8.1. Across all datasets, audio waveforms are preprocessed to 16kHz using `torchaudio`.

Beethoven. The dataset consists of recordings of Beethoven’s 32 piano sonatas. We use the version of the dataset shared by Mehri et al. [135], which can be found [here](#). Since we compare to numbers reported by Mehri et al. [135], we use linear quantization for all (and only) Beethoven experiments. We attempt to match the splits used by the original paper by reference to the code provided [here](#).

YouTubeMix. A 4 hour dataset of piano music taken from https://www.youtube.com/watch?v=Eh0_MrRfftU. We split the audio track into .wav files of 1 minute each, and use the first 88% files for training, next 6% files for validation and final 6% files for testing.

SC09. The Speech Commands dataset [222] contains many spoken words by thousands of speakers under various recording conditions including some very noisy environments. Following prior work [49, 112] we use the subset that contains spoken digits “zero” through “nine”. This SC09 dataset contains 31,158 training utterances (8.7 hours in total) by 2,032 speakers, where each audio has length 1 second sampled at 16kHz. the generative models need to model them without any conditional information.

The datasets we used can be found on Huggingface datasets: [Beethoven](#), [YouTubeMix](#), [SC09](#).

H.2.2 Models and Training Details

For all datasets, SASHIMI, SampleRNN and WaveNet receive 8-bit quantized inputs. During training, we use no additional data augmentation of any kind. We summarize the hyperparameters used and any sweeps performed for each method below.

Details for Autoregressive Models

All methods in the AR setting were trained on single V100 GPU machines.

SaShiMi. The S4 block inside the SASHIMI architecture follows the same general model described in Section 7.2. The models for some experiments differ in small details because they were trained before the complete S4 parameterization was consolidated. For example, the Hurwitz parameterization in Section 3.3.3 was actually developed in the course of the

paper this chapter is based on, and not all experiments were re-run afterwards. Some of the details that differ include:

- Some experiments only trained Λ and C with the recommended learning rate of 0.001, and freeze all other parameters for simplicity (including P, B, Δ).
- Some experiments implement the weight-tying of Section 7.2.2, although even then only for the Λ parameter since the others were frozen.
- The GLU variant (Section 7.2.3) was only used on SC09.

Updating the training details can likely improve performance; see for example the footnote in Table 8.6.

For the overall SASHIMI architecture, we use feature expansion of $2\times$ when pooling, and use a feedforward dimension of $2\times$ the model dimension in all inverted bottlenecks in the model. We use a model dimension of 64. We train with $4\times \rightarrow 4\times$ pooling for all datasets, with 8 S4 blocks per tier.

We train SASHIMI on Beethoven for 1M steps, YouTubeMix for 600K steps, SC09 for 1.1M steps.

SampleRNN. We adapt an [open-source PyTorch implementation](#) of the SampleRNN backbone, and train it using truncated backpropagation through time (TBPTT) with a chunk size of 1024. We train 2 variants of SampleRNN: a 3-tier model with frame sizes 8, 2, 2 with 1 RNN per layer to match the 3-tier RNN from Mehri et al. [135] and a 2-tier model with frame sizes 16, 4 with 2 RNNs per layer that we found had stronger performance in our replication (than the 2-tier model from Mehri et al. [135]). For the recurrent layer, we use a standard GRU model with orthogonal weight initialization following Mehri et al. [135], with hidden dimension 1024 and feedforward dimension 256 between tiers. We also use weight normalization as recommended by Mehri et al. [135].

We train SampleRNN on Beethoven for 150K steps, YouTubeMix for 200K steps, SC09 for 300K steps. We found that SampleRNN could be quite unstable, improving steadily and then suddenly diverging. It also appeared to be better suited to training with linear rather than mu-law quantization.

WaveNet. We adapt an [open-source PyTorch implementation](#) of the WaveNet backbone, trained using standard backpropagation. We set the number of residual channels to 64,

dilation channels to 64, end channels to 512. We use 4 blocks of dilation with 10 layers each, with a kernel size of 2. Across all datasets, we sweep the number of skip channels among {512, 1024}. For optimization, we use the AdamW optimizer, with a learning rate of 0.001 and a plateau learning rate scheduler that has a patience of 5 on the validation NLL. During training, we use a batch size of 1 and pad each batch on the left with zeros equal to the size of the receptive field of the WaveNet model (4093 in our case).

We train WaveNet on Beethoven for 400K steps, YouTubeMix for 200K steps, SC09 for 500K steps.

Details for Diffusion Models

All diffusion models were trained on 8-GPU A100 machines.

DiffWave. We adapt an [open-source PyTorch implementation](#) of the DiffWave model. The DiffWave baseline in Table 8.6 is the unconditional SC09 model reported in Kong et al. [112], which uses a 36 layer WaveNet backbone with dilation cycle

$$[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048]$$

and hidden dimension 256, a linear diffusion schedule $\beta_t \in [1 \times 10^4, 0.02]$ with $T = 200$ steps, and the Adam optimizer with learning rate 2×10^{-4} . The small DiffWave model reported in Table 8.7 has 30 layers with dilation cycle $[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]$ and hidden dimension 128.

DiffWave with SaShiMi. Our large SASHIMI model has hidden dimension 128 and 6 S4 blocks per tier with the standard two pooling layers with pooling factor 4 and expansion factor 2 (Section 8.3). We additionally have S4 layers in the down-blocks in addition to the up-blocks of Figure 8.1. Our small SASHIMI model (Table 8.7) reduces the hidden dimension to 64. These architectures were chosen to roughly parameter match the DiffWave model. While DiffWave experimented with other architectures such as deep and thin WaveNets or different dilation cycles [112], we only ran a single SASHIMI model of each size. All optimization and diffusion hyperparameters were kept the same, with the exception that we manually decayed the learning rate of the large SASHIMI model at 500K steps as it had saturated and the model had already caught up to the best DiffWave model (Table 8.7).

H.2.3 Automated Evaluations

NLL. We report negative log-likelihood (NLL) scores for all AR models in bits, on the test set of the respective datasets. To evaluate NLL, we follow the same protocol as we would for training, splitting the data into non-overlapping chunks (with the same length as training), running each chunk through a model and then using the predictions made on each step of that chunk to calculate the average NLL for the chunk.

Evaluation of generated samples. Following Kong et al. [112], we use 4 standard evaluation metrics for generative models evaluated using an auxiliary ResNeXT classifier [228] which achieved 98.3% accuracy on the test set. Note that Kong et al. [112] reported an additional metric NDB (number of statistically-distinct bins), which we found to be slow to compute and generally uninformative, despite SASHIMI performing best.

- **Fréchet Inception Distance (FID)** [84] uses the classifier to compare moments of generated and real samples in feature space.
- **Inception Score (IS)** [181] measures both quality and diversity of generated samples, and favoring samples that the classifier is confident on.
- **Modified Inception Score (mIS)** [77] provides a measure of both intra-class in addition to inter-class diversity.
- **AM Score** [243] uses the marginal label distribution of training data compared to IS.

We also report the Cohen’s inter-annotator agreement κ score, which is computed with the classifier as one rater and a crowdworker’s digit prediction as the other rater (treating the set of crowdworkers as a single rater).

Evaluation Procedure for Autoregressive Models

Because autoregressive models have tractable likelihood scores that are easily evaluated, we use them to perform a form of rejection sampling when evaluating their automated metrics. Each model generated 5120 samples and ranked them by likelihood scores. The lowest-scoring 0.40 and highest-scoring 0.05 fraction of samples were thrown out. The remaining samples were used to calculate the automated metrics.

The two thresholds for the low- and high- cutoffs were found by validation on a separate set of 5120 generated samples.

Figure H.3: (**YouTubeMix MOS interface.**) Crowdsourcing interface for collecting mean opinion scores (MOS) on YouTubeMix. Crowdworkers are given a collection of audio files, one from each method and the dataset. They are asked to rate each file on audio fidelity and musicality.

Rate the audio fidelity and musicality of piano music.

Please use headphones in a quiet environment if possible.

Some files may be loud, so we recommend keeping volumes at a moderate level.

You will be presented a batch of recordings and asked to rate each of them on audio fidelity and musicality.

Some are computer generated, while others are performed by a human.

Fidelity: How clear is the audio? Does it sound like it's coming from a walkie-talkie (bad fidelity) or a studio-quality sound system (excellent fidelity)?

Musicality: To what extent does the recording sound like real piano music? Does it change in unusual ways (bad musicality) or is it musically consistent (excellent musicality)?

Feel free to listen to each recording as many times as you like and update your scores as you compare the methods.

	Fidelity					Musicality				
	1: Bad Very noisy audio	2: Poor Mostly noisy audio	3: Fair Somewhat clear audio	4: Good Mostly clear audio	5: Excellent Clear audio	1: Not at all Not musical at all	2: Slightly Somewhat musical	3: Moderately Moderately musical	4: Very Very musical	5: Extremely Extremely musical
▶ 0:00 / 0:16	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
▶ 0:00 / 0:16	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
▶ 0:00 / 0:16	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
▶ 0:00 / 0:16	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
▶ 0:00 / 0:16	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Submit										

Evaluation Procedure for Non-autoregressive Models

Automated metrics were calculated on 2048 random samples generated from each model.

H.2.4 Evaluation of Mean Opinion Scores

For evaluating mean opinion scores (MOS), we repurpose scripts for creating jobs for Amazon Mechanical Turk from Neekhara et al. [142].

Mean Opinion Scores for YouTubeMix

We collect MOS scores on audio fidelity and musicality, following Dieleman, Oord, and Simonyan [47]. The instructions and interface used are shown in Figure H.3.

The protocol we follow to collect MOS scores for YouTubeMix is outlined below. For this study, we compare unconditional AR models, SAShIMI to SampleRNN and WaveNet.

- For each method, we generated unconditional 1024 samples, where each sample had length 16s (1.024M steps). For sampling, we directly sample from the distribution

Figure H.4: (**SC09 MOS interface**.) Crowdsourcing interface for collecting mean opinion scores (MOS) on SC09. Crowdworkers are given a collection of 10 audio files from the same method, and are asked to classify the spoken digits and rate them on intelligibility. At the bottom, they provide a single score on the audio quality and speaker diversity they perceive for the batch.

Rate and annotate audio files containing spoken digits.

Please use headphones in a quiet environment if possible. Read the instructions below carefully before starting the task.

You are presented a batch of recordings and asked to classify what digit you hear in each of them. If you are unsure which digit it is, select the one that sounds most like the recording to you.

You are also asked to rate the intelligibility of each recording

Intelligibility: How easily could you identify the recorded digits? Are they impossible to classify (not at all intelligible) or very easy to understand (extremely intelligible)?

Intelligibility: How easily could you identify the recorded digits? Are they clear enough to be understood?

Think about the recordings you heard when answering these questions.

Think about the recordings you heard when answering these questions.

Digit Classification										
	0 Zero	1 One	2 Two	3 Three	4 Four	5 Five	6 Six	7 Seven	8 Eight	9 Nine
▶ 0.00 / 0.00 ————— ♦ :	<input type="radio"/>									
▶ 0.00 / 0.01 ————— ♦ :	<input type="radio"/>									
▶ 0.00 / 0.01 ————— ⚡ :	<input type="radio"/>									
▶ 0.00 / 0.01 ————— ♦ :	<input type="radio"/>									
▶ 0.00 / 0.01 ————— ♦ :	<input type="radio"/>									

Audio Quality					Speaker Diversity				
1: Bad	2: Poor	3: Fair	4: Good	5: Excellent	1: Not at all	2: Slightly	3: Moderately	4: Very	5: Extremely
Very noisy audio	Mouthy noisy audio	Somewhat clear audio	Mostly clear audio	Clear audio	Not at all diverse (none or no distinct speakers)	Slightly diverse (few distinct speakers)	Moderately diverse (many distinct speakers)	Very diverse (almost all distinct speakers)	Extremely diverse (all distinct speakers)
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>					

Digit Intelligibility									
1: Not at all Not at all intelligible	2: Slightly Slightly intelligible	3: Moderately Moderately intelligible	4: Very Very intelligible	5: Extremely Extremely intelligible					
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>					
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>					
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>					
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>					
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>					

output by the model at each time step, without using any other modifications.

- As noted by Mehri et al. [135], autoregressive models can sometimes generate samples that are “noise-like”. To fairly compare all methods, we sequentially inspect the samples and reject any that are noise-like. We also remove samples that mostly consist of silences (defined as more than half the clip being silence). We carry out this process until we have 30 samples per method.
- Next, we randomly sample 25 clips from the dataset. Since this evaluation is quite subjective, we include some gold standard samples. We add 4 clips that consist mostly of noise (and should have musicality and quality MOS ≤ 2). We include 1 clip that has variable quality but musicality MOS ≤ 2 . Any workers who disagree with this assessment have their responses omitted from the final evaluation.
- We construct 30 batches, where each batch consists of 1 sample per method (plus a single sample for the dataset), presented in random order to a crowdworker. We use Amazon Mechanical Turk for collecting responses, paying \$0.50 per batch and collecting 20 responses per batch. We use Master qualifications for workers, and restrict to workers with a HIT approval rating above 98%. We note that it is likely enough to collect 10 responses per batch.

Mean Opinion Scores for SC09

Next, we outline the protocol used for collecting MOS scores on SC09. We collect MOS scores on digit intelligibility, audio quality and speaker diversity, as well as asking crowdworkers to classify digits following Donahue, McAuley, and Puckette [49]. The instructions and interface used are shown in Figure H.4.

- For each method, we generate 2048 samples of 1s each. For autoregressive models (SASHIMI, SampleRNN, WaveNet), we directly sample from the distribution output by the model at each time step, without any modification. For WaveGAN, we obtained 50000 randomly generated samples from the authors, and subsampled 2048 samples randomly from this set. For the diffusion models, we run 200 steps of denoising following Kong et al. [112].
- We use the ResNeXT model (Appendix H.2.3) to classify the generated samples into digit categories. Within each digit category, we choose the top-50 samples, as ranked by classifier confidence. We note that this mimics the protocol followed by Donahue,

McAuley, and Puckette [49], which we established through correspondence with the authors.

- Next, we construct batches consisting of 10 random samples (randomized over all digits) drawn from a single method (or the dataset). Each method (and the dataset) thus has 50 total batches. We use Amazon Mechanical Turk for collecting responses, paying \$0.20 per batch and collecting 10 responses per batch. We use Master qualification for workers, and restrict to workers with a HIT approval rating above 98%.

Note that we elicit digit classes and digit intelligibility scores for each audio file, while audio quality and speaker diversity are elicited once per batch.

Appendix I

Appendix for Chapter 9

I.1 Theory Details

We formalize and generalize the notation of Section 9.3.1 and prove the results.

For the remainder of this section, we fix a dimension D (e.g. $D = 2$ for a 2-dimensional SSM). The D -dimensional SSM will be a map from a function $u : \mathbb{R}^D \rightarrow \mathbb{C}$ to $y : \mathbb{R}^D \rightarrow \mathbb{C}$.

Definition I.1 (Indexing notation). *Let $[d]$ denote the set $\{1, 2, \dots, d\}$. Let $[0]$ denote the empty set $\{\}$. Given a subset $I \subseteq [D]$, let $-I$ denote its complement $[D] \setminus I$. Let $[-d] = -[d] = \{d + 1, \dots, D\}$.*

Definition I.2 (Tensor products and contractions). *Given $a \in \mathbb{C}^{N_1}$ and $b \in \mathbb{C}^{N_2}$, let $a \otimes b \in \mathbb{C}^{N_1 \times N_2}$ be defined as $(a \otimes b)_{n_1, n_2} = a_{n_1} b_{n_2}$.*

Given a tensor $x \in \mathbb{C}^{N_1 \times \dots \times N_D}$ and matrix $\mathbf{A} \in \mathbb{C}^{N_1 \times N_1}$, define $\mathbf{A} \cdot^{(1)} x \in \mathbb{C}^{N_1 \times \dots \times N_D}$ as

$$(\mathbf{A} \cdot^{(1)} x)_{n_1, \dots, n_D} = \sum_m \mathbf{A}_{n_1 m} x_{k, n_2, \dots, n_D}.$$

which is simply a matrix multiplication over the first dimension. Let $\cdot^{(\tau)}$ similarly denote matrix multiplication over any other axis.

It will be easier to work directly from the convolutional definition of SSM (equation (9.2)). We will then show equivalence to a PDE formulation (equation (9.1)).

Definition I.3 (Multidimensional SSM). Let N_1, \dots, N_D be state sizes for each of the D dimensions. The D -dimensional SSM has parameters $\mathbf{A}^{(\tau)} \in \mathbb{C}^{N^{(\tau)} \times N^{(\tau)}}$, $\mathbf{B}^{(\tau)} \in \mathbb{C}^{N^{(\tau)}}$, and $\mathbf{C} \in \mathbb{C}^{N^{(1)} \times \dots \times N^{(D)}}$ for each dimension $\tau \in [D]$.

and is defined by the map $u \mapsto y$ given by

$$\begin{aligned} x(t) &= (u * \bigotimes_{\tau=1}^D e^{t^{(\tau)} \mathbf{A}^{(\tau)}} \mathbf{B}^{(\tau)})(t) \\ y(t) &= \langle \mathbf{C}, x(t) \rangle. \end{aligned}$$

Note that at all times $t = (t^{(1)}, \dots, t^{(D)})$, the dimension of the state is $x(t) \in \mathbb{C}^{N_1 \times \dots \times N_D}$.

Finally, it will be convenient for us to define ‘‘partial bindings’’ of u, x, y where one or more of the coordinates are fixed.

Definition I.4 (Partial binding of u). Let $I \subseteq [D]$ and let $t^I = (t^{(I_1)}, \dots, t^{(I_{|I|})})$ be an partial assignment of the time variables. Define $u_I(t^I)$ to be the function $\mathbb{R}^{D-|I|} \mapsto \mathbb{C}$ where the I indices are fixed to t^I .

For example, if $D = 3$ and $I = [1]$, then

$$u_I(t^I) = u(t^{(1)}, \cdot, \cdot)$$

is a function from $\mathbb{R}^2 \rightarrow \mathbb{C}$ mapping $(t^{(2)}, t^{(3)}) \mapsto u(t^{(1)}, t^{(2)}, t^{(3)})$.

Definition I.5 (Partial binding of x).

$$x_I(t^I)(t^{-I}) = (u_I(t^I) * \bigotimes_{\tau \in -I} e^{t^{(\tau)} \mathbf{A}^{(\tau)}} \mathbf{B}^{(\tau)})(t^{-I})$$

Note that $x_{[0]}() : \mathbb{R}^D \mapsto \mathbb{C}^{N_1 \times \dots \times N_D}$ coincides with the full state x .

The following more formal theorem shows the equivalence of this convolutional LTI system with a standard 1D SSM differential equation in each dimension.

Theorem I.6. Given a partial state $x_I(t^I)$ and a time variable $t^{(\tau)}$ for $\tau \notin I$, let $I' = I \cup \{\tau\}$. Then the partial derivatives satisfy

$$\begin{aligned} \frac{\partial x_I(t^I)}{\partial t^{(\tau)}}(t^{-I}) &= \mathbf{A}^{(\tau)} \cdot (\tau) x_I(t^I)(t^{-I}) \\ &\quad + \mathbf{B}^{(\tau)} \otimes x_{I'}(t^{I'})(t^{-I'}) \end{aligned}$$

Proof. WLOG assume $I = [d]$, since all notions are permutation-independent. We will consider differentiating the state x_I with respect to the time variable $t^{(d+1)}$. The key fact is that differentiating a convolution $\frac{d}{dt}(f*g)$ is equivalent to differentiating one of the operands $f * (\frac{d}{dt}g)$:

$$\begin{aligned}
& \frac{\partial x_{[d]}(t^{[d]})}{\partial t^{(d+1)}}(t^{-[d]}) \\
&= \frac{\partial}{\partial t^{(d+1)}}(u_I(t^I) * \bigotimes_{\tau \in -I} e^{t^{(\tau)} \mathbf{A}^{(\tau)}} \mathbf{B}^{(\tau)})(t^{-I}) \\
&= (u_I(t^I) * \frac{\partial}{\partial t^{(d+1)}} \bigotimes_{\tau \in -[d]} e^{t^{(\tau)} \mathbf{A}^{(\tau)}} \mathbf{B}^{(\tau)})(t^{-I}) \\
&= \left(u_I(t^I) * \left(\mathbf{A}^{(d+1)} e^{t^{(d+1)} \mathbf{A}^{(d+1)}} \mathbf{B}^{(d+1)} + \mathbf{B}^{(d+1)} \delta(t^{(d+1)}) \right) \bigotimes_{\tau \in -[d+1]} e^{t^{(\tau)} \mathbf{A}^{(\tau)}} \mathbf{B}^{(\tau)} \right) (t^{-I}) \\
&= \mathbf{A}^{(d+1) \cdot (d+1)} \left(u_I(t^I) * \bigotimes_{\tau \in -[d]} e^{t^{(\tau)} \mathbf{A}^{(\tau)}} \mathbf{B}^{(\tau)} \right) (t^{-I}) \\
&\quad + u_{[d+1]}(t^{[d+1]}) \mathbf{B}^{(d+1)} \bigotimes_{\tau \in -[d+1]} e^{t^{(\tau)} \mathbf{A}^{(\tau)}} \mathbf{B}^{(d+1)} \\
&= \mathbf{A}^{(d+1) \cdot (d+1)} x_{[d]}(t^{[d]})(t^{-[d]}) \\
&\quad + \mathbf{B}^{(d+1)} \otimes x_{[d+1]}(t^{[d+1]})(t^{-[d+1]}).
\end{aligned}$$

□

The following corollary follows immediately from linearity of the convolution operator, allowing the order of convolution and inner product by \mathbf{C} to be switched.

Corollary I.7. *The output y is equivalent to $y = K * u$ where*

$$K(t) = \langle \mathbf{C}, \bigotimes_{\tau=1}^D e^{t^{(\tau)} \mathbf{A}^{(\tau)}} \mathbf{B}^{(\tau)} \rangle$$

This completes the proof of Theorem 9.2.

Finally, Corollary 9.3 is an immediate consequence of the Kronecker mixed-product identity

$$(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC}) \otimes (\mathbf{BD}).$$

I.2 Experimental Details

We use PyTorch for all experiments, and build on the publicly available S4 code.

I.2.1 Image Classification

ImageNet training: all models were trained from scratch with no outside data using 8 Nvidia A100 GPUs. For both ViT and ConvNeXt experiments, we follow the procedure from T2T-ViT [231] and the original ConvNeXt [131], respectively, with minor adjustments. Preprocessing and dataloading was done using the TIMM [223] library. In S4ND-ViT, we turn off weight decay and remove the class token prepending of the input sequence. For the ConvNeXt models, we add RepeatAug [88], as well as reduce the batch size to 3840 for S4ND-ConvNeXt.

S4ND specific settings include a bidirectional S4 kernel (Section 7.2.6), and a state dimension of $N = 64$ for the SSMs.

I.2.2 Video Classification

HMDB-51 training: all models were trained from scratch on a single Nvidia A100 GPU. We use the Pytorchvideo library for data loading and minimal preprocessing of RGB frames only (no optical flow). During training, we randomly sample 2 second clips from each video. At validation and test time, 2 second clips are sampled uniformly to ensure that an entire video is seen. For each ConvNeXt 3D video model (baselines and S4ND versions), we fix the hyperparameters in Table I.2, and do a sweep over the RandAugment[34] settings `num_layers= {1, 2}` and `magnitude= {3, 5, 7, 9}`.

S4ND specific settings were similar to image classification, a bidirectional S4 kernel and a state dimension of 64 for the SSMs.

Table I.1: (**Hyperparameters for image classification.**) ImageNet settings for ViT and ConvNeXt baseline models.

	ViT	CONVNEXT
image size	224 ²	224 ²
optimizer	AdamW	AdamW
optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.999$	$\beta_1, \beta_2 = 0.9, 0.999$
weight init	trunc. normal (std=0.02)	trunc. normal (std=0.02)
base learning rate	0.001	0.004
weight decay	0.05	0.05
dropout	None	None
batch size	4096	4096
training epochs	300	300
learning rate schedule	cosine decay	cosine decay
warmup epochs	10	20
warmup schedule	linear	linear
layer-wise lr decay [14, 32]	None	None
randaugment [34]	(9,0.5,layers=2)	(9,0.5,layers=2)
mixup [237]	0.8	0.8
cutmix [232]	1.0	1.0
repeataug [88]	None	3
random erasing [240]	0.25	0.25
label smoothing [196]	0.1	0.1
stochastic depth [91]	0.1	0.1
layer scale [208]	None	1e-6
head init scale [208]	None	None
exp.mov. avg (EMA) [164]	0.9999	None

I.2.3 Continuous Time Capabilities Experiments

We use the CIFAR-10 and Celeb-A datasets for all our experiments. Below, we include all experimental details including data processing, model training and hyperparameters, and evaluation details.

As noted in Table 9.3, we consider 3 resolutions for each dataset, a base resolution that is considered the standard resolution for that dataset, as well as two lower resolutions low and mid. Images at the lower resolutions are generated by taking an image at the base resolution, and then downsampling using the `resize` function in `torchvision`, with bilinear interpolation and antialiasing turned on.

CIFAR-10. The base resolution is chosen to be 32×32 , which is the resolution at which models are generally trained on this dataset. The lower resolutions are 16×16 and 8×8 . We use no data augmentation for either zero-shot resolution change or progressive resizing

Table I.2: (**Hyperparameters for video classification.**) HMDB-51 settings for all 3D ConvNeXt models (baselines and S4ND models).

CONVNEXT 3D	
image size	224 ²
# frames	30
clip duration (sec)	2
optimizer	AdamW
optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.999$
weight init	trunc. normal (std=0.02)
base learning rate	0.0001
weight decay	0.2
dropout (head)	0.2
batch size	64
training epochs	50
learning rate schedule	cosine decay
warmup epochs	0
stochastic depth [91]	0.2
layer scale [208]	1e-6
head init scale [208]	None
exp.mov. avg (EMA) [164]	None

experiments. We train with standard cross-entropy loss on the task of 10-way classification. For reporting results, we use standard accuracy over the 10 classes.

We use a simple isotropic model backbone, identical to the one in Section 7.2 for the (1D) S4 model. The only difference is that we use either the S4ND or Conv2D layer in place of the S4 block, which ensures that the model can accept a batch of 2D spatial inputs with channels. For results reported in the main body, we use a 6×256 architecture consisting of 6 layers and a model dimension of 256.

For all CIFAR-10 experiments, we train for 100 epochs, use a base learning rate of 0.01 and a weight decay of 0.03.

Celeb-A. On Celeb-A, we consider 160×160 as the base resolution, and 128×128 and 64×64 as the lower resolutions. Images on Celeb-A are generally 218×178 , and we run a `CenterCrop(178) → Resize(160)` transform in `torchvision` to resize the image. We use no data augmentation for either zero-shot resolution change or progressive resizing experiments. We train with standard binary cross-entropy loss on the task of 40-way multilabel attribute classification on Celeb-A. For reporting results, we use the multilabel binary accuracy, i.e. the average binary accuracy across all 40 tasks.

Similar to our ImageNet experiments, we use a ConvNeXt model as the basic backbone for experiments on Celeb-A. However, use a particularly small model here, consisting of 4 stages with depths $(3, 3, 3, 3)$ and corresponding model dimensions $(64, 128, 256, 512)$. For S4ND, we simply follow the same process as ImageNet, replacing all depthwise Conv2D layers with S4ND. A minor difference is that we use a global convolution in the stem downsampling for S4ND, rather than the smaller kernel size that we use for ImageNet. We use a drop path rate of 0.1 for both S4ND and the Conv2D baseline in all experiments.

For all Celeb-A experiments, we train for 20 epochs, use a base learning rate of 0.004, and use automatic mixed precision for training.

Other fixed hyperparameters. For all experiments, we use bidirectional S4 kernels (Section 7.2.6), and use a state dimension of $N = 64$ for the S4 layers. We use AdamW as the optimizer.

Zero-Shot Resolution Change

For Table 9.4, we simply train on a single resolution among base, mid and low, and then directly test at all higher resolutions. For model selection at a particular test resolution, we select the best performing model for that test resolution i.e. we checkpoint the model at the epoch where it performs best at the test resolution in question. Note that we refer to “test resolutions” but report validation metrics in Table 9.4, as is standard practice for additional experiments and ablations.

CIFAR-10. We use a batch size of 50, and for both methods (and all resolutions), we use a cosine decay schedule for the learning rate with no restarts and a length of 100000, and a linear warmup of 500 steps.

For baseline Conv2D hyperparameters, we compare the performance of Conv2D with and without depthwise convolutions. Otherwise, all hyperparameters are fixed to the common values laid out in the previous section (and are identical to those used for S4ND).

For S4ND hyperparameters, we sweep the choice of initialization for the state space parameters A, B in the S4 model among {S4-LegS, S4-FouT, S4D-Lin, S4D-Inv} (Table 6.1) and sweep bandlimit (α) values among $\{0.05, 0.10, 0.20, 0.50, \infty\}$.

Celeb-A. We use a batch size of 256 when training on the lower resolutions, and 128 when training at the base resolution. For both methods (and all resolutions), we use a cosine decay schedule for the learning rate with no restarts and a length of 13000 (batch size 256)

or 26000 (batch size 128), and a linear warmup of 500 steps.

For baseline ConvNeXt hyperparameters, we use a weight decay of 0.1 after an initial sweep over weight decay values $\{0.05, 0.1, 0.2, 0.5\}$.

For S4ND hyperparameters, we sweep the choice of initialization for the state space parameters \mathbf{A}, \mathbf{B} in the S4 model among {S4-LegS, S4-FouT, S4D-Lin, S4D-Inv}, sweep bandlimit (α) values among $\{0.05, 0.10, 0.20, 0.50, \infty\}$, and use a weight decay of 1.0. The value of the weight decay was chosen based on an initial exploratory sweep over weight decay values $\{0.1, 0.2, 0.5, 1.0, 5.0\}$.

Progressive Resizing

For Table 9.5, we train with a resizing schedule that progressively increases the resolution of the data. We report all metrics on validation sets, similar to our zero-shot experiments. We only use 2 resizing stages in our experiments, as we found in preliminary experiments that 2+ stages had little to no benefit.

All hyperparameters and training details are identical to the zero-shot resolution change experiments, except for those we describe next. When training with multiple stages, we reset the cosine learning rate scheduler at the beginning of each stage (along with the linear warmup). The length of the scheduler is varied in proportion to the length of the stage, e.g. on CIFAR-10, for a stage 50 epochs long, we change the length of the decay schedule from 100000 steps to 50000 steps (and similarly for Celeb-A). We always use a linear warmup of 500 steps for each stage. Additionally, Table 9.5 denotes the length (in epochs) of each stage, as well as their training resolutions.

Another important detail is that we set the bandlimit parameter carefully for each stage. We found that bandlimiting in the first stage is critical to final performance, and without bandlimiting the performance on the base resolution is substantially worse. Bandlimiting is generally not useful for a stage if it is training at the base resolution i.e. at the resolution that we will be testing at (in our experiments, this is the case only for second stage training on CIFAR-10).

On CIFAR-10, we use a bandlimit α of 0.10 or 0.20 for the first stage, depending on whether the first stage was training at 8×8 (low) or 16×16 (mid) respectively. In the second stage, we always train at the base resolution and use no bandlimiting, since as stated earlier, it has a negligible effect on performance.

For Celeb-A, we set the bandlimit parameter to 0.1 for both stages. Note that on Celeb-A, we do not train at the base resolution at all, only training at low in the first stage and mid in the second stage.

Bibliography

- [1] Waqar Ahmad, Hazrat Ali, Zubair Shah, and Shoaib Azmat. “A new generative adversarial network for medical images super resolution”. In: *Scientific Reports* 12.1 (2022), p. 9533.
- [2] Hassan Akbari, Liangzhe Yuan, Rui Qian, Wei-Hong Chuang, Shih-Fu Chang, Yin Cui, and Boqing Gong. “Vatt: Transformers for multimodal self-supervised learning from raw video, audio and text”. In: *Advances in Neural Information Processing Systems* 34 (2021).
- [3] Juan Miguel Lopez Alcaraz and Nils Strodthoff. “Diffusion-based time series imputation and forecasting with structured state space models”. In: *arXiv preprint arXiv:2208.09399* (2022).
- [4] George B Arfken and Hans J Weber. *Mathematical methods for physicists*. Elsevier Academic Press, 2005.
- [5] Martin Arjovsky, Amar Shah, and Yoshua Bengio. “Unitary evolution recurrent neural networks”. In: *The International Conference on Machine Learning (ICML)*. 2016, pp. 1120–1128.
- [6] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. “Vivit: A video vision transformer”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 6836–6846.
- [7] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. “Layer normalization”. In: *arXiv preprint arXiv:1607.06450* (2016).
- [8] Alexei Baevski and Michael Auli. “Adaptive input representations for neural language modeling”. In: *arXiv preprint arXiv:1809.10853* (2018).
- [9] Anthony Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn Keogh. “The UEA multivariate time series classification archive, 2018”. In: *arXiv preprint arXiv:1811.00075* (2018).

- [10] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014).
- [11] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling”. In: *arXiv preprint arXiv:1803.01271* (2018).
- [12] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. “Trellis networks for sequence modeling”. In: *The International Conference on Learning Representations (ICLR)*. 2019.
- [13] George A Baker, George A Baker Jr, Peter Graves-Morris, and Susan S Baker. *Pade Approximants: Encyclopedia of Mathematics and It's Applications, Vol. 59 George A. Baker, Jr., Peter Graves-Morris*. Vol. 59. Cambridge University Press, 1996.
- [14] Hangbo Bao, Li Dong, and Furu Wei. “Beit: Bert pre-training of image transformers”. In: *arXiv preprint arXiv:2106.08254* (2021).
- [15] Mikołaj Bińkowski, Jeff Donahue, Sander Dieleman, Aidan Clark, Erich Elsen, Norman Casagrande, Luis C Cobo, and Karen Simonyan. “High Fidelity Speech Synthesis with Adversarial Networks”. In: *International Conference on Learning Representations*. 2020.
- [16] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. “On the opportunities and risks of foundation models”. In: *arXiv preprint arXiv:2108.07258* (2021).
- [17] James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. “Quasi-recurrent neural networks”. In: *arXiv preprint arXiv:1611.01576* (2016).
- [18] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.
- [19] John Charles Butcher and Nicolette Goodwin. *Numerical methods for ordinary differential equations*. Vol. 2. Wiley Online Library, 2008.
- [20] Joao Carreira and Andrew Zisserman. “Quo vadis, action recognition? a new model and the kinetics dataset”. In: *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6299–6308.
- [21] Chaw-Bing Chang and Michael Athans. “State estimation for discrete systems with switching parameters”. In: *IEEE Transactions on Aerospace and Electronic Systems* 3 (1978), pp. 418–425.

- [22] Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael Witbrock, Mark Hasegawa-Johnson, and Thomas S Huang. “Dilated recurrent neural networks”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.
- [23] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. “Recurrent neural networks for multivariate time series with missing values”. In: *Scientific reports* 8.1 (2018), pp. 1–12.
- [24] Nanxin Chen, Yu Zhang, Heiga Zen, Ron J Weiss, Mohammad Norouzi, and William Chan. “WaveGrad: Estimating Gradients for Waveform Generation”. In: *International Conference on Learning Representations*. 2021.
- [25] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. “Neural ordinary differential equations”. In: *Advances in neural information processing systems*. 2018, pp. 6571–6583.
- [26] T. S. Chihara. *An introduction to orthogonal polynomials*. Dover Books on Mathematics. Dover Publications, 2011. ISBN: 9780486479293.
- [27] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. “Generating long sequences with sparse transformers”. In: *arXiv preprint arXiv:1904.10509* (2019).
- [28] Narsimha Chilkuri and Chris Eliasmith. “Parallelizing Legendre Memory Unit Training”. In: *The International Conference on Machine Learning (ICML)* (2021).
- [29] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014.
- [30] Krzysztof Choromanski, Valerii Likhoshesterov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. “Rethinking attention with performers”. In: *The International Conference on Learning Representations (ICLR)*. 2020.
- [31] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: *arXiv preprint arXiv:1412.3555* (2014).
- [32] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. “Electra: Pre-training text encoders as discriminators rather than generators”. In: *arXiv preprint arXiv:2003.10555* (2020).
- [33] Taco Cohen and Max Welling. “Group equivariant convolutional networks”. In: *International conference on machine learning*. PMLR. 2016, pp. 2990–2999.

- [34] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. “Randaugment: Practical automated data augmentation with a reduced search space”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 702–703.
- [35] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Sanja Fidler, Antonino Furnari, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, et al. “Scaling egocentric vision: The epic-kitchens dataset”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 720–736.
- [36] Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. “FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness”. In: *arXiv preprint arXiv:2205.14135* (2022).
- [37] Tri Dao, Daniel Y Fu, Khaled K Saab, Armin W Thomas, Atri Rudra, and Christopher Ré. “Hungry Hungry Hippos: Towards Language Modeling with State Space Models”. In: *The International Conference on Learning Representations (ICLR)*. 2023.
- [38] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. “Language modeling with gated convolutional networks”. In: *International conference on machine learning*. PMLR. 2017, pp. 933–941.
- [39] Jared Quincy Davis, Albert Gu, Tri Dao, Krzysztof Choromanski, Christopher Ré, Percy Liang, and Chelsea Finn. “Catformer: Designing Stable Transformers via Sensitivity Analysis”. In: *The International Conference on Machine Learning (ICML)*. 2021.
- [40] Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau. “GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.
- [41] Christopher De Sa, Albert Gu, Rohan Puttagunta, Christopher Ré, and Atri Rudra. “A two-pronged progress in structured dense matrix vector multiplication”. In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2018, pp. 1060–1079.
- [42] DeepSound. *SampleRNN*. <https://github.com/deepsound-project/samplernn-pytorch>. 2017.
- [43] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

- [44] Arjun D Desai, Beliz Gunel, Batu M Ozturkler, Harris Beg, Shreyas Vasanawala, Brian A Hargreaves, Christopher Ré, John M Pauly, and Akshay S Chaudhari. “VORTEX: Physics-Driven Data Augmentations for Consistency Training for Robust Accelerated MRI Reconstruction”. In: *arXiv preprint arXiv:2111.02549* (2021).
- [45] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. “Jukebox: A generative model for music”. In: *arXiv preprint arXiv:2005.00341* (2020).
- [46] Sander Dieleman, Aäron van den Oord, and Karen Simonyan. “The challenge of realistic music generation: modelling raw audio at scale”. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018, pp. 8000–8010.
- [47] Sander Dieleman, Aäron van den Oord, and Karen Simonyan. “The challenge of realistic music generation: modelling raw audio at scale”. In: *ArXiv abs/1806.10474* (2018).
- [48] Mingyu Ding, Bin Xiao, Noel Codella, Ping Luo, Jingdong Wang, and Lu Yuan. “DaViT: Dual Attention Vision Transformers”. In: *arXiv preprint arXiv:2204.03645* (2022).
- [49] Chris Donahue, Julian McAuley, and Miller Puckette. “Adversarial Audio Synthesis”. In: *International Conference on Learning Representations*. 2019.
- [50] Chris Donahue, Julian McAuley, and Miller Puckette. “Adversarial Audio Synthesis”. In: *ICLR*. 2019.
- [51] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *arXiv preprint arXiv:2010.11929* (2020).
- [52] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: <http://archive.ics.uci.edu/ml>.
- [53] Y. Eidelman and I. Gohberg. “On a new class of structured matrices”. In: *Integral Equations and Operator Theory* 34 (1999).
- [54] Jesse Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts. “DDSP: Differentiable digital signal processing”. In: *arXiv preprint arXiv:2001.04643* (2020).
- [55] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Mohammad Norouzi, Douglas Eck, and Karen Simonyan. “Neural audio synthesis of musical notes with wavenet autoencoders”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1068–1077.

- [56] N Benjamin Erichson, Omri Azencot, Alejandro Queiruga, Liam Hodgkinson, and Michael W Mahoney. “Lipschitz recurrent neural networks”. In: *International Conference on Learning Representations*. 2021.
- [57] *Training imagenet in 3 hours for 25 minutes*. <https://www.fast.ai/2018/04/30/dawnbench-fastai/>. 2018.
- [58] Yassir Fathullah, Chunyang Wu, Yuan Shangguan, Junteng Jia, Wenhan Xiong, Jay Mahadeokar, Chunxi Liu, Yangyang Shi, Ozlem Kalinli, Mike Seltzer, et al. “Multi-Head State Space Model for Sequence Modeling”. In: (2023).
- [59] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. “Slowfast networks for video recognition”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 6202–6211.
- [60] Marc Finzi, Samuel Stanton, Pavel Izmailov, and Andrew Gordon Wilson. “Generalizing Convolutional Neural Networks for Equivariance to Lie Groups on Arbitrary Continuous Data”. In: *ICML*. 2020.
- [61] Emily Fox, Erik Sudderth, Michael Jordan, and Alan Willsky. “Nonparametric Bayesian learning of switching linear dynamical systems”. In: *Advances in neural information processing systems* 21 (2008).
- [62] Emily B Fox, Erik B Sudderth, Michael I Jordan, and Alan S Willsky. “A sticky HDP-HMM with application to speaker diarization”. In: *The Annals of Applied Statistics* (2011), pp. 1020–1056.
- [63] Karl J Friston, Lee Harrison, and Will Penny. “Dynamic causal modelling”. In: *Neuroimage* 19.4 (2003), pp. 1273–1302.
- [64] Daniel Y Fu, Elliot L Epstein, Eric Nguyen, Armin W Thomas, Michael Zhang, Tri Dao, Atri Rudra, and Christopher Ré. “Simple hardware-efficient long convolutions for sequence modeling”. In: *arXiv preprint arXiv:2302.06646* (2023).
- [65] Zoubin Ghahramani and Geoffrey E Hinton. “Variational learning for switching state-space models”. In: *Neural computation* 12.4 (2000), pp. 831–864.
- [66] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.
- [67] Karan Goel, Albert Gu, Chris Donahue, and Christopher Ré. “It’s Raw! Audio Generation with State-Space Models”. In: *The International Conference on Machine Learning (ICML)*. 2022.

- [68] Gene H Golub and Charles F Van Loan. *Matrix computations*. Vol. 3. JHU press, 2013.
- [69] Raghav Goyal, Samira Ebrahimi Kahou, Vincent Michalski, Joanna Materzynska, Susanne Westphal, Heuna Kim, Valentin Haenel, Ingo Fruend, Peter Yianilos, Moritz Mueller-Freitag, et al. “The “something something” video database for learning and evaluating visual common sense”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 5842–5850.
- [70] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. “HiPPO: Recurrent Memory with Optimal Polynomial Projections”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020.
- [71] Albert Gu, Karan Goel, and Christopher Ré. “Efficiently Modeling Long Sequences with Structured State Spaces”. In: *The International Conference on Learning Representations (ICLR)*. 2022.
- [72] Albert Gu, Caglar Gulcehre, Tom Le Paine, Matt Hoffman, and Razvan Pascanu. “Improving the Gating Mechanism of Recurrent Neural Networks”. In: *The International Conference on Machine Learning (ICML)*. 2020.
- [73] Albert Gu, Ankit Gupta, Karan Goel, and Christopher Ré. “On the Parameterization and Initialization of Diagonal State Space Models”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2022.
- [74] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. “Combining recurrent, convolutional, and continuous-time models with the structured learnable linear state space layer”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021.
- [75] Albert Gu, Isys Johnson, Aman Timalsina, Atri Rudra, and Christopher Ré. “How to Train Your HiPPO: State Space Models with Generalized Basis Projections”. In: *The International Conference on Learning Representations (ICLR)*. 2023.
- [76] Ankit Gupta. “Diagonal State Spaces are as Effective as Structured State Spaces”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2022.
- [77] Swaminathan Gurumurthy, Ravi Kiran Sarvadevabhatla, and R Venkatesh Babu. “Deligan: Generative adversarial networks for diverse and limited data”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 166–174.
- [78] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. “Learning spatio-temporal features with 3d residual networks for action recognition”. In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017, pp. 3154–3160.

- [79] Ramin Hasani, Mathias Lechner, Tsun-Hsuan Wang, Makram Chahine, Alexander Amini, and Daniela Rus. “Liquid structural state-space models”. In: *The International Conference on Learning Representations (ICLR)*. 2023.
- [80] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. arXiv: [1502.01852 \[cs.CV\]](https://arxiv.org/abs/1502.01852).
- [81] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
- [82] Dan Hendrycks and Kevin Gimpel. “Gaussian error linear units (gelus)”. In: *arXiv preprint arXiv:1606.08415* (2016).
- [83] Dan Hendrycks, Norman Mu, Ekin D Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. “AugMix: A simple data processing method to improve robustness and uncertainty”. In: *arXiv preprint arXiv:1912.02781* (2019).
- [84] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. “Gans trained by a two time-scale update rule converge to a local nash equilibrium”. In: *Advances in neural information processing systems* 30 (2017).
- [85] Sepp Hochreiter. “Untersuchungen zu dynamischen neuronalen Netzen”. In: *Diploma, Technische Universität München* 91.1 (1991).
- [86] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.
- [87] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.
- [88] Elad Hoffer, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoefer, and Daniel Soudry. “Augment your batch: Improving generalization through instance repetition”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 8129–8138.
- [89] Elad Hoffer, Berry Weinstein, Itay Hubara, Tal Ben-Nun, Torsten Hoefer, and Daniel Soudry. “Mix & match: training convnets with mixed image sizes for improved accuracy, speed and scale resiliency”. In: *arXiv preprint arXiv:1908.08986* (2019).
- [90] Sara Hooker. “The hardware lottery”. In: *Communications of the ACM* 64.12 (2021), pp. 58–65.
- [91] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. “Deep networks with stochastic depth”. In: *European conference on computer vision*. Springer. 2016, pp. 646–661.

- [92] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.
- [93] Md Mohaiminul Islam and Gedas Bertasius. “Long movie clip classification with state-space video models”. In: *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXV*. Springer. 2022, pp. 87–104.
- [94] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. “Deep learning for time series classification: a review”. In: *Data mining and knowledge discovery* 33.4 (2019), pp. 917–963.
- [95] Herbert Jaeger and Harald Haas. “Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication”. In: *Science* 304.5667 (2004), pp. 78–80.
- [96] Vivek Jayaram and John Thickstun. “Parallel and flexible sampling from autoregressive models via langevin dynamics”. In: *The International Conference on Machine Learning (ICML)*. 2021.
- [97] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. “3D convolutional neural networks for human action recognition”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.1 (2012), pp. 221–231.
- [98] Matthew James Johnson and Alan S Willsky. “Bayesian nonparametric hidden semi-Markov models”. In: (2013).
- [99] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. “An empirical exploration of recurrent network architectures”. In: *International Conference on Machine Learning*. 2015, pp. 2342–2350.
- [100] Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aäron van den Oord, Sander Dieleman, and Koray Kavukcuoglu. “Efficient neural audio synthesis”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 2410–2419.
- [101] Rudolph Emil Kalman. “A new approach to linear filtering and prediction problems”. In: (1960).
- [102] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. “Recurrent experience replay in distributed reinforcement learning”. In: *International conference on learning representations*. 2019.
- [103] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. “Large-scale video classification with convolutional neural

- networks”. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2014, pp. 1725–1732.
- [104] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. “Transformers are rnns: Fast autoregressive transformers with linear attention”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5156–5165.
- [105] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. “The kinetics human action video dataset”. In: *arXiv preprint arXiv:1705.06950* (2017).
- [106] Patrick Kidger. “On neural differential equations”. In: *arXiv preprint arXiv:2202.02435* (2022).
- [107] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. “Neural Controlled Differential Equations for Irregular Time Series”. In: *arXiv preprint arXiv:2005.08926* (2020).
- [108] Sungwon Kim, Sang-Gil Lee, Jongyoon Song, Jaehyeon Kim, and Sungroh Yoon. “FloWaveNet: A Generative Flow for Raw Audio”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 3370–3378.
- [109] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *The International Conference on Learning Representations (ICLR)*. 2015.
- [110] W Bastiaan Kleijn, Felicia SC Lim, Alejandro Luebs, Jan Skoglund, Florian Stimberg, Quan Wang, and Thomas C Walters. “Wavenet based low rate speech coding”. In: *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2018, pp. 676–680.
- [111] Dan Kondratyuk, Liangzhe Yuan, Yandong Li, Li Zhang, Mingxing Tan, Matthew Brown, and Boqing Gong. “Movinets: Mobile video networks for efficient video recognition”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 16020–16030.
- [112] Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. “DiffWave: A Versatile Diffusion Model for Audio Synthesis”. In: *International Conference on Learning Representations*. 2021.
- [113] Thomas William Körner. *Fourier analysis*. Cambridge university press, 1989.
- [114] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: *Citeseer* (2009).
- [115] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

- [116] Hildegard Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso Poggio, and Thomas Serre. “HMDB: a large video database for human motion recognition”. In: *2011 International conference on computer vision*. IEEE. 2011, pp. 2556–2563.
- [117] Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brébisson, Yoshua Bengio, and Aaron C Courville. “MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis”. In: *Advances in Neural Information Processing Systems* 32 (2019).
- [118] Kushal Lakhota, Evgeny Kharitonov, Wei-Ning Hsu, Yossi Adi, Adam Polyak, Benjamin Bolte, Tu-Anh Nguyen, Jade Copet, Alexei Baevski, Adelrahman Mohamed, et al. “Generative spoken language modeling from raw audio”. In: *arXiv preprint arXiv:2102.01192* (2021).
- [119] Tao Lei. “When attention meets fast recurrence: Training language models with reduced compute”. In: *arXiv preprint arXiv:2102.12459* (2021).
- [120] Tao Lei, Yu Zhang, Sida I Wang, Hui Dai, and Yoav Artzi. “Simple recurrent units for highly parallelizable recurrence”. In: *arXiv preprint arXiv:1709.02755* (2017).
- [121] Mario Lezcano-Casado and David Martínez-Rubio. “Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group”. In: *The International Conference on Machine Learning (ICML)*. 2019.
- [122] Naihan Li, Shujie Liu, Yanqing Liu, Sheng Zhao, and Ming Liu. “Neural speech synthesis with transformer network”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 6706–6713.
- [123] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. “Independently recurrent neural network (IndRNN): Building a longer and deeper RNN”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5457–5466.
- [124] Yuhong Li, Tianle Cai, Yi Zhang, Deming Chen, and Debadeepta Dey. “What Makes Convolutional Models Great on Long Sequence Modeling?” In: *The International Conference on Learning Representations (ICLR)*. 2023.
- [125] Scott Linderman, Matthew Johnson, Andrew Miller, Ryan Adams, David Blei, and Liam Paninski. “Bayesian learning and inference in recurrent switching linear dynamical systems”. In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 914–922.
- [126] Vasileios Lioutas and Yuhong Guo. “Time-aware large kernel convolutions”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 6172–6183.

- [127] Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. “Understanding the difficulty of training transformers”. In: *The International Conference on Machine Learning (ICML)* (2020).
- [128] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, et al. “Swin Transformer V2: Scaling Up Capacity and Resolution”. In: *arXiv preprint arXiv:2111.09883* (2021).
- [129] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. “Swin transformer: Hierarchical vision transformer using shifted windows”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 10012–10022.
- [130] Ze Liu, Jia Ning, Yue Cao, Yixuan Wei, Zheng Zhang, Stephen Lin, and Han Hu. “Video swin transformer”. In: *arXiv preprint arXiv:2106.13230* (2021).
- [131] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. “A convnet for the 2020s”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 11976–11986.
- [132] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. “Deep Learning Face Attributes in the Wild.” In: *ICCV*. IEEE Computer Society, 2015, pp. 3730–3738. ISBN: 978-1-4673-8391-2. URL: <http://dblp.uni-trier.de/db/conf/iccv/iccv2015.html#LiuLWT15>.
- [133] Ilya Loshchilov and Frank Hutter. “Decoupled weight decay regularization”. In: *The International Conference on Learning Representations (ICLR)*. 2019.
- [134] Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, and Luke Zettlemoyer. “Mega: moving average equipped gated attention”. In: *The International Conference on Learning Representations (ICLR)*. 2023.
- [135] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. “SampleRNN: An Unconditional End-to-End Neural Audio Generation Model”. In: *International Conference on Learning Representations*. 2017.
- [136] Harsh Mehta, Ankit Gupta, Ashok Cutkosky, and Behnam Neyshabur. “Long range language modeling via gated state spaces”. In: *The International Conference on Learning Representations (ICLR)*. 2023.
- [137] Stephen Merity, Nitish Shirish Keskar, James Bradbury, and Richard Socher. “Scalable Language Modeling: WikiText-103 on a Single GPU in 12 hours”. In: *SysML* (2018).

- [138] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. “Nerf: Representing scenes as neural radiance fields for view synthesis”. In: *European conference on computer vision*. Springer. 2020, pp. 405–421.
- [139] Mathew Monfort, Alex Andonian, Bolei Zhou, Kandan Ramakrishnan, Sarah Adel Bargal, Tom Yan, Lisa Brown, Quanfu Fan, Dan Gutfreund, Carl Vondrick, et al. “Moments in time dataset: one million videos for event understanding”. In: *IEEE transactions on pattern analysis and machine intelligence* 42.2 (2019), pp. 502–508.
- [140] James Morrill, Christopher Salvi, Patrick Kidger, James Foster, and Terry Lyons. “Neural Rough Differential Equations for Long Time Series”. In: *The International Conference on Machine Learning (ICML)* (2021).
- [141] Kevin P Murphy. “Switching kalman filters”. In: (1998).
- [142] Paarth Neekhara, Chris Donahue, Miller Puckette, Shlomo Dubnov, and Julian McAuley. “Expediting TTS Synthesis with Adversarial Vocoding”. In: *INTERSPEECH*. 2019.
- [143] Eric Nguyen, Karan Goel, Albert Gu, Gordon Downs, Preey Shah, Tri Dao, Stephen Baccus, and Christopher Ré. “S4nd: Modeling images and videos as multidimensional signals with state spaces”. In: *Advances in Neural Information Processing Systems*. 2022.
- [144] Naoki Nonaka and Jun Seita. “In-depth benchmarking of deep neural network architectures for ECG diagnosis”. In: *Machine Learning for Healthcare Conference*. PMLR. 2021, pp. 414–439.
- [145] Bruno A Olshausen and David J Field. “Natural image statistics and efficient coding”. In: *Network: computation in neural systems* 7.2 (1996), p. 333.
- [146] Aäron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George Driessche, Edward Lockhart, Luis Cobo, Florian Stimberg, et al. “Parallel wavenet: Fast high-fidelity speech synthesis”. In: *International conference on machine learning*. PMLR. 2018, pp. 3918–3926.
- [147] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. “Neural discrete representation learning”. In: *arXiv preprint arXiv:1711.00937* (2017).
- [148] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. “Wavenet: A generative model for raw audio”. In: *arXiv preprint arXiv:1609.03499* (2016).

- [149] OpenAI. “GPT-4 Technical Report”. In: *CoRR* abs/2303.08774 (2023). DOI: [10.48550/arXiv.2303.08774](https://doi.org/10.48550/48550/arXiv.2303.08774). arXiv: [2303.08774](https://arxiv.org/abs/2303.08774). URL: <https://doi.org/10.48550/arXiv.2303.08774>.
- [150] Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. “Resurrecting recurrent neural networks for long sequences”. In: *arXiv preprint arXiv:2303.06349* (2023).
- [151] Tom Le Paine, Pooya Khorrami, Shiyu Chang, Yang Zhang, Prajit Ramachandran, Mark A Hasegawa-Johnson, and Thomas S Huang. “Fast Wavenet Generation Algorithm”. In: *arXiv preprint arXiv:1611.09482* (2016).
- [152] Victor Pan. *Structured matrices and polynomials: unified superfast algorithms*. Springer Science & Business Media, 2001.
- [153] Victor Pan. “Fast approximate computations with Cauchy matrices and polynomials”. In: *Mathematics of Computation* 86.308 (2017), pp. 2799–2826.
- [154] Victor Y Pan. “Transformations of matrix structures work again”. In: *Linear Algebra and Its Applications* 465 (2015), pp. 107–138.
- [155] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. “Image transformer”. In: *International conference on machine learning*. PMLR. 2018, pp. 4055–4064.
- [156] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks”. In: *International conference on machine learning*. 2013, pp. 1310–1318.
- [157] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems* 32 (2019).
- [158] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, et al. “RWKV: Reinventing RNNs for the Transformer Era”. In: *arXiv preprint arXiv:2305.13048* (2023).
- [159] Kainan Peng, Wei Ping, Zhao Song, and Kexin Zhao. “Non-autoregressive neural text-to-speech”. In: *International conference on machine learning*. PMLR. 2020, pp. 7586–7598.
- [160] Clément Pernet. “Computing with quasiseparable matrices”. In: *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*. 2016, pp. 389–396.

- [161] Wei Ping, Kainan Peng, and Jitong Chen. “ClariNet: Parallel Wave Generation in End-to-End Text-to-Speech”. In: *International Conference on Learning Representations*. 2019.
- [162] Wei Ping, Kainan Peng, Kexin Zhao, and Zhao Song. “WaveFlow: A compact flow-based model for raw audio”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 7706–7716.
- [163] Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. “Hyena hierarchy: Towards larger convolutional language models”. In: *arXiv preprint arXiv:2302.10866* (2023).
- [164] Boris T Polyak and Anatoli B Juditsky. “Acceleration of stochastic approximation by averaging”. In: *SIAM journal on control and optimization* 30.4 (1992), pp. 838–855.
- [165] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. “Waveglow: A flow-based generative network for speech synthesis”. In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 3617–3621.
- [166] John G Proakis. *Digital signal processing: principles algorithms and applications*. Pearson Education India, 2001.
- [167] Zhaofan Qiu, Ting Yao, and Tao Mei. “Learning spatio-temporal representation with pseudo-3d residual networks”. In: *proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 5533–5541.
- [168] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. *Language Models are Unsupervised Multitask Learners*. 2019.
- [169] Jack Rae, Chris Dyer, Peter Dayan, and Timothy Lillicrap. “Fast Parametric Learning with Activation Memorization”. In: *The International Conference on Machine Learning (ICML)* (2018).
- [170] Prajit Ramachandran, Tom Le Paine, Pooya Khorrami, Mohammad Babaeizadeh, Shiyu Chang, Yang Zhang, Mark A Hasegawa-Johnson, Roy H Campbell, and Thomas S Huang. “Fast generation for convolutional autoregressive models”. In: *arXiv preprint arXiv:1704.06001* (2017).
- [171] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. “Zero-shot text-to-image generation”. In: *arXiv preprint arXiv:2102.12092* (2021).
- [172] M. Ravanelli, T. Parcollet, and Y. Bengio. “The PyTorch-Kaldi Speech Recognition Toolkit”. In: *In Proc. of ICASSP*. 2019.

- [173] David W Romero, Robert-Jan Bruintjes, Jakub M Tomczak, Erik J Bekkers, Mark Hoogendoorn, and Jan C van Gemert. “Flexconv: Continuous kernel convolutions with differentiable kernel sizes”. In: *The International Conference on Learning Representations (ICLR)*. 2022.
- [174] David W Romero, Anna Kuzina, Erik J Bekkers, Jakub M Tomczak, and Mark Hoogendoorn. “CKConv: Continuous Kernel Convolution For Sequential Data”. In: *arXiv preprint arXiv:2102.02611* (2021).
- [175] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5–9, 2015, Proceedings, Part III* 18. Springer. 2015, pp. 234–241.
- [176] Yulia Rubanova, Tian Qi Chen, and David K Duvenaud. “Latent Ordinary Differential Equations for Irregularly-Sampled Time Series”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 5321–5331.
- [177] Daniel Ruderman and William Bialek. “Statistics of natural images: Scaling in the woods”. In: *Advances in neural information processing systems* 6 (1993).
- [178] T Konstantin Rusch and Siddhartha Mishra. “UnICORNN: A recurrent model for learning very long time dependencies”. In: *The International Conference on Machine Learning (ICML)* (2021).
- [179] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision* 115 (2015), pp. 211–252.
- [180] Khaled Saab, Jared Dunnmon, Christopher Ré, Daniel Rubin, and Christopher Lee-Messer. “Weak supervision as an efficient approach for automated seizure detection in electroencephalography”. In: *NPJ Digital Medicine* 3.1 (2020), pp. 1–12.
- [181] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. “Improved techniques for training gans”. In: *Advances in neural information processing systems* 29 (2016), pp. 2234–2242.
- [182] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. “Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications”. In: *arXiv preprint arXiv:1701.05517* (2017).
- [183] George Saon, Ankit Gupta, and Xiaodong Cui. “Diagonal State Space Augmented Transformers for Speech Recognition”. In: *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing* (2023), pp. 1–5.

- Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2023, pp. 1–5.
- [184] Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. “SchNet: A continuous-filter convolutional neural network for modeling quantum interactions”. In: *NeurIPS*. 2017.
 - [185] Vinit Shah, Eva Von Weltin, Silvia Lopez, James Riley McHugh, Lillian Veloso, Meysam Golmohammadi, Iyad Obeid, and Joseph Picone. “The Temple University hospital seizure detection corpus”. In: *Frontiers in neuroinformatics* 12 (2018), p. 83.
 - [186] Noam Shazeer. “GLU variants improve transformer”. In: *arXiv preprint arXiv:2002.05202* (2020).
 - [187] Jonathan Shen, Ruoming Pang, Ron J Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, et al. “Natural tts synthesis by conditioning wavenet on mel spectrogram predictions”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 4779–4783.
 - [188] Jiaxin Shi, Ke Alexander Wang, and Emily B Fox. “Sequence Modeling with Multiresolution Convolutional Memory”. In: *arXiv preprint arXiv:2305.01638* (2023).
 - [189] Victor Shoup. *A computational introduction to number theory and algebra*. Cambridge university press, 2009.
 - [190] Gunnar A Sigurdsson, Güл Varol, Xiaolong Wang, Ali Farhadi, Ivan Laptev, and Abhinav Gupta. “Hollywood in homes: Crowdsourcing data collection for activity understanding”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 510–526.
 - [191] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).
 - [192] Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. “Simplified State Space Layers for Sequence Modeling”. In: *The International Conference on Learning Representations (ICLR)*. 2023.
 - [193] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.
 - [194] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. “Sequence to sequence learning with neural networks”. In: *Advances in neural information processing systems* 27 (2014).
 - [195] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. “Going

- deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.
- [196] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.
- [197] G. Szegö. *Orthogonal Polynomials*. American Mathematical Society colloquium publications v.23. American Mathematical Society, 1967. ISBN: 9780821889527.
- [198] Corentin Tallec and Yann Ollivier. “Can recurrent neural networks warp time?” In: *The International Conference on Learning Representations (ICLR)*. 2018.
- [199] Chang Wei Tan, Christoph Bergmeir, Francois Petitjean, and Geoffrey I Webb. “Time Series Extrinsic Regression”. In: *Data Mining and Knowledge Discovery* (2021), pp. 1–29. DOI: <https://doi.org/10.1007/s10618-021-00745-9>.
- [200] Mingxing Tan and Quoc Le. “Efficientnetv2: Smaller models and faster training”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 10096–10106.
- [201] Siyi Tang, Jared Dunnmon, Liangqiong Qu, Khaled Kamal Saab, Tina Baykaner, Christopher Lee-Messer, and Daniel Rubin. “Modeling Multivariate Biosignals with Graph Neural Networks and Structured State Space”. In: *ICLR 2023 Workshop on Time Series Representation Learning for Health*. 2023.
- [202] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. “Long Range Arena : A Benchmark for Efficient Transformers”. In: *International Conference on Learning Representations*. 2021. URL: <https://openreview.net/forum?id=qVyeW-grC2k>.
- [203] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. “Efficient transformers: A survey”. In: *ACM Computing Surveys* 55.6 (2022), pp. 1–28.
- [204] Yee Teh, Michael Jordan, Matthew Beal, and David Blei. “Sharing clusters among related groups: Hierarchical Dirichlet processes”. In: *Advances in neural information processing systems* 17 (2004).
- [205] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, et al. “Mlp-mixer: An all-mlp architecture for vision”. In: *arXiv preprint arXiv:2105.01601* (2021).
- [206] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. “Efficient object localization using convolutional networks. CoRR abs/1411.4280 (2014)”. In: *arXiv preprint arXiv:1411.4280* (2014).

- [207] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. “Training data-efficient image transformers & distillation through attention”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 10347–10357.
- [208] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. “Going deeper with image transformers”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 32–42.
- [209] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Hervé Jégou. “Fixing the train-test resolution discrepancy”. In: *Advances in neural information processing systems* 32 (2019).
- [210] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. “Learning spatiotemporal features with 3d convolutional networks”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4489–4497.
- [211] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. “A closer look at spatiotemporal convolutions for action recognition”. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2018, pp. 6450–6459.
- [212] Lloyd N Trefethen. *Approximation theory and approximation practice*. Vol. 164. SIAM, 2019.
- [213] Trieu H Trinh, Andrew M Dai, Minh-Thang Luong, and Quoc V Le. “Learning longer-term dependencies in RNNs with auxiliary losses”. In: *The International Conference on Machine Learning (ICML)*. 2018.
- [214] Arnold Tustin. “A method of analysing the behaviour of linear systems in terms of time series”. In: *Journal of the Institution of Electrical Engineers-Part IIA: Automatic Regulators and Servo Mechanisms* 94.1 (1947), pp. 130–142.
- [215] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. “Conditional image generation with pixelcnn decoders”. In: *Advances in neural information processing systems* 29 (2016).
- [216] Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. “Pixel recurrent neural networks”. In: *International conference on machine learning*. PMLR. 2016, pp. 1747–1756.
- [217] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention Is All You Need”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

- [218] Aaron Voelker, Ivana Kajić, and Chris Eliasmith. “Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 15544–15553.
- [219] Aaron Russell Voelker. “Dynamical systems in spiking neuromorphic hardware”. PhD thesis. University of Waterloo, 2019.
- [220] Junxiong Wang, Jing Nathan Yan, Albert Gu, and Alexander M Rush. “Pretraining Without Attention”. In: *arXiv preprint arXiv:2212.10544* (2022).
- [221] Xin Wang, Shinji Takaki, and Junichi Yamagishi. “Neural source-filter-based waveform model for statistical parametric speech synthesis”. In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 5916–5920.
- [222] Pete Warden. “Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition”. In: *ArXiv* abs/1804.03209 (2018).
- [223] Ross Wightman. *PyTorch Image Models*. <https://github.com/rwightman/pytorch-image-models>. 2019. doi: [10.5281/zenodo.4414861](https://doi.org/10.5281/zenodo.4414861).
- [224] Robert L Williams, Douglas A Lawrence, et al. *Linear state-space control systems*. Wiley Online Library, 2007.
- [225] Max A Woodbury. “Inverting modified matrices”. In: *Memorandum report* 42 (1950), p. 106.
- [226] Chao-Yuan Wu and Philipp Krahenbuhl. “Towards long-form video understanding”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 1884–1894.
- [227] Felix Wu, Angela Fan, Alexei Baevski, Yann N Dauphin, and Michael Auli. “Pay less attention with lightweight and dynamic convolutions”. In: *The International Conference on Learning Representations (ICLR)*. 2019.
- [228] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. “Aggregated residual transformations for deep neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1492–1500.
- [229] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. “Rethinking spatiotemporal feature learning for video understanding”. In: *arXiv preprint arXiv:1712.04851* 1.2 (2017), p. 5.
- [230] Ryuichi Yamamoto, Eunwoo Song, and Jae-Min Kim. “Parallel WaveGAN: A fast waveform generation model based on generative adversarial networks with multi-resolution spectrogram”. In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 6199–6203.

- [231] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. “Tokens-to-token vit: Training vision transformers from scratch on imagenet”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 558–567.
- [232] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. “Cutmix: Regularization strategy to train strong classifiers with localizable features”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 6023–6032.
- [233] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. “Are transformers effective for time series forecasting?” In: *arXiv preprint arXiv:2205.13504* (2022).
- [234] Shuangfei Zhai, Walter Talbott, Nitish Srivastava, Chen Huang, Hanlin Goh, Ruixiang Zhang, and Josh Susskind. “An attention free transformer”. In: *arXiv preprint arXiv:2105.14103* (2021).
- [235] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. “Scaling Vision Transformers”. In: *CoRR* abs/2106.04560 (2021).
- [236] Guofeng Zhang, Tongwen Chen, and Xiang Chen. “Performance recovery in digital implementation of analogue systems”. In: *SIAM journal on control and optimization* 45.6 (2007), pp. 2207–2223.
- [237] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. “mixup: Beyond empirical risk minimization”. In: *arXiv preprint arXiv:1710.09412* (2017).
- [238] Jiong Zhang, Yibo Lin, Zhao Song, and Inderjit S Dhillon. “Learning long term dependencies via Fourier recurrent units”. In: *The International Conference on Machine Learning (ICML)*. 2018.
- [239] Michael Zhang, Khaled K Saab, Michael Poli, Tri Dao, Karan Goel, and Christopher Ré. “Effectively modeling time series with simple discrete state spaces”. In: *The International Conference on Learning Representations (ICLR)*. 2023.
- [240] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. “Random erasing data augmentation”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 2020, pp. 13001–13008.
- [241] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. “Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting”. In: *The Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Conference*. Vol. 35. 12. AAAI Press, 2021, pp. 11106–11115.

- [242] Linqi Zhou, Michael Poli, Winnie Xu, Stefano Massaroli, and Stefano Ermon. “Deep Latent State Space Models for Time-Series Generation”. In: *arXiv preprint arXiv:2212.12749* (2022).
- [243] Zhiming Zhou, Han Cai, Shu Rong, Yuxuan Song, Kan Ren, Weinan Zhang, Jun Wang, and Yong Yu. “Activation Maximization Generative Adversarial Nets”. In: *International Conference on Learning Representations*. 2018.
- [244] Simiao Zuo, Xiaodong Liu, Jian Jiao, Denis Charles, Eren Manavoglu, Tuo Zhao, and Jianfeng Gao. “Efficient Long Sequence Modeling via State Space Augmented Transformer”. In: *arXiv preprint arXiv:2212.08136* (2022).

Albert Gu

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Christopher Ré) Principal Advisor

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Percy Liang)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Scott Linderman)

Approved for the Stanford University Committee on Graduate Studies
