
HiPPO: Recurrent Memory with Optimal Polynomial Projections

Albert Gu^{*†}, Tri Dao^{*†}, Stefano Ermon[†], Atri Rudra[‡], Christopher Ré[†]

[†] Department of Computer Science, Stanford University

[‡] Department of Computer Science and Engineering, University at Buffalo, SUNY

{albertgu, trid}@stanford.edu, ermon@cs.stanford.edu, atri@buffalo.edu, chrismre@cs.stanford.edu

Abstract

A central problem in learning from sequential data is representing cumulative history in an incremental fashion as more data is processed. We introduce a general framework (HiPPO) for the online compression of continuous signals and discrete time series by projection onto polynomial bases. Given a measure that specifies the importance of each time step in the past, HiPPO produces an optimal solution to a natural *online function approximation* problem. As special cases, our framework yields a short derivation of the recent Legendre Memory Unit (LMU) from first principles, and generalizes the ubiquitous gating mechanism of recurrent neural networks such as GRUs. This formal framework yields a new memory update mechanism (HiPPO-LegS) that scales through time to remember all history, avoiding priors on the timescale. HiPPO-LegS enjoys the theoretical benefits of timescale robustness, fast updates, and bounded gradients. By incorporating the memory dynamics into recurrent neural networks, HiPPO RNNs can empirically capture complex temporal dependencies. On the benchmark permuted MNIST dataset, HiPPO-LegS sets a new state-of-the-art accuracy of 98.3%. Finally, on a novel trajectory classification task testing robustness to out-of-distribution timescales and missing data, HiPPO-LegS outperforms RNN and neural ODE baselines by 25-40% accuracy.

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 [34, 38, 56] (e.g., the “vanishing gradients” problem). Although various heuristics have been proposed to overcome this, such as gates in the successful LSTM and GRU [16, 34], or higher-order frequencies in the recent Fourier Recurrent Unit [79] and Legendre Memory Unit (LMU) [71], 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 [66, 71]; this can be problematic in settings with distribution shift (e.g. arising from different instrument sampling rates in medical data [62, 63]). Finally, many of them lack theoretical guarantees on how well they capture long-term dependencies, such as gradient bounds. To design

*Equal contribution. Order determined by coin flip.

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 [14]. With their rich and well-studied history [65], along with their widespread use in approximation theory [68] and signal processing [57], 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 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 [71] (Section 2.3), which proposes an update rule that projects onto fixed-length sliding windows through time.² 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 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 3).

We integrate the HiPPO memory mechanisms into RNNs, and empirically show that they outperform baselines on standard tasks used to benchmark long-term dependencies. On the permuted MNIST dataset, our hyperparameter-free HiPPO-LegS method achieves a new state-of-the-art accuracy of 98.3%, beating the previous RNN SoTA by over 1 point and even outperforming models with global context such as transformers (Section 4.1). Next, we 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 4.2). Finally, we validate HiPPO’s theory, including computational efficiency and scalability, allowing fast and accurate online function reconstruction over millions of time steps (Section 4.3). Code for reproducing our experiments is available at <https://github.com/HazyResearch/hippo-code>.

2 The HiPPO Framework: High-order Polynomial Projection Operators

We motivate the problem of online function approximation with projections as an approach to learning memory representations (Section 2.1). Section 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 2.3 instantiates the framework to recover the LMU and yield new memory updates (e.g. HiPPO-LagT), demonstrating the generality of the HiPPO framework. Section 2.4 discusses how to convert the main continuous-time results into practical discrete versions. Finally in Section 2.5 we show how gating in RNNs is an instance of HiPPO memory.

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

²The LMU was originally motivated by spiking neural networks in modeling biological nervous systems; its derivation is not self-contained but a sketch can be pieced together from [71, 72, 73].

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 $(e^{2\pi ix})^n$ (cf. Appendix D.4). In Appendix C, 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 $t \rightarrow \infty$.

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 1) departs from prior work on sliding transforms in several ways, which we discuss in detail in Appendix A.1. 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 C 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 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 $\|f_{\leq t} - g^{(t)}\|_{L_2(\mu^{(t)})}$.
- (2) $\text{coef}_t : \mathcal{G} \rightarrow \mathbb{R}^N$ 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 $\mu^{(t)}$.

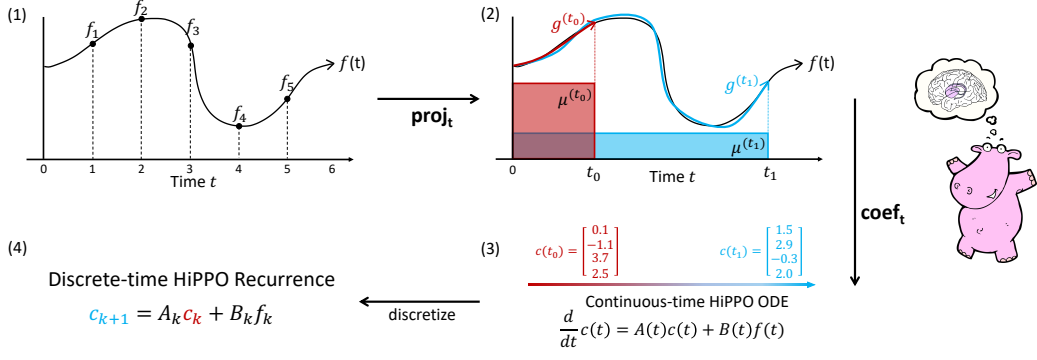


Figure 1: **Illustration of the HiPPO framework.** (1) For any function f , (2) at every time t there is an optimal projection $g^{(t)}$ of f onto the space of polynomials, with respect to a measure $\mu^{(t)}$ weighing the past. (3) For an appropriately chosen basis, the corresponding coefficients $c(t) \in \mathbb{R}^N$ 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 $(f_k)_{k \in \mathbb{N}}$.

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) will show that the coefficient function $c(t) = \text{coef}_t(\text{proj}_t(f))$ has the form of an ODE satisfying $\frac{d}{dt} c(t) = A(t)c(t) + B(t)f(t)$ for some $A(t) \in \mathbb{R}^{N \times N}$, $B(t) \in \mathbb{R}^{N \times 1}$. 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 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.

2.3 High Order Projection: Measure Families and HiPPO ODEs

Our main theoretical results are instantiations of HiPPO for various measure families $\mu^{(t)}$. We provide two examples of natural sliding window measures and the corresponding projection operators. The unified perspective on memory mechanisms allows us to derive these closed-form solutions with the same strategy, provided in Appendices D.1, D.2. The first explains the core Legendre Memory Unit (LMU) [71] update in a principled way and characterizes its limitations, while the other is novel, demonstrating the generality of the HiPPO framework. Appendix D contrasts the tradeoffs of these measures (Fig. 5), contains proofs of their derivations, and derives additional HiPPO formulas for other bases such as Fourier (recovering the Fourier Recurrent Unit [79]) and Chebyshev.

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. The **translated Laguerre (LagT)** measures instead use the exponentially decaying measure, assigning more importance to recent history.

$$\text{LegT: } \mu^{(t)}(x) = \frac{1}{\theta} \mathbb{I}_{[t-\theta, t]}(x) \quad \text{LagT: } \mu^{(t)}(x) = e^{-(t-x)} \mathbb{I}_{(-\infty, t]}(x) = \begin{cases} e^{x-t} & \text{if } x \leq t \\ 0 & \text{if } x > t \end{cases}$$

Theorem 1. For LegT and LagT, the hippo operators satisfying Definition 1 are given by linear time-invariant (LTI) ODEs $\frac{d}{dt} c(t) = -Ac(t) + Bf(t)$, where $A \in \mathbb{R}^{N \times N}$, $B \in \mathbb{R}^{N \times 1}$:

$$\text{LegT: } A_{nk} = \frac{1}{\theta} \begin{cases} (-1)^{n-k} (2n+1) & \text{if } n \geq k \\ 2n+1 & \text{if } n < k \end{cases}, \quad B_n = \frac{1}{\theta} (2n+1) (-1)^n \quad \text{LagT: } A_{nk} = \begin{cases} 1 & \text{if } n \geq k \\ 0 & \text{if } n < k \end{cases}, \quad B_n = 1 \quad (1)$$

Equation (1) proves the LMU update [71, equation (1)]. Additionally, our derivation (Appendix D.1) 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.

2.4 HiPPO recurrences: from Continuous to Discrete Time with ODE Discretization

Since actual data is inherently discrete (e.g. sequences and time series), we discuss how the HiPPO projection operators can be discretized using standard techniques, so that the continuous-time HiPPO ODEs 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 $f(k \cdot \Delta t) = f_k$ for some step size Δt , (iii) produces a function $c(t)$ through the ODE dynamics, and (iv) discretizes back to an output sequence $c_k := c(k \cdot \Delta t)$.

The basic method of discretizing an ODE $\frac{d}{dt}c(t) = u(t, c(t), f(t))$ chooses a step size Δt and performs the discrete updates $c(t + \Delta t) = c(t) + \Delta t \cdot u(t, c(t), f(t))$.³ In general, this process is sensitive to the *discretization step size* hyperparameter Δt .

Finally, we note that this provides a way to seamlessly handle timestamped data, even with missing values: the difference between timestamps indicates the (adaptive) Δt to use in discretization [13]. Appendix B.3 contains a full discussion of discretization.

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 discretized version of HiPPO-LagT (2) becomes $c(t + \Delta t) = c(t) + \Delta t(-Ac(t) + Bf(t)) = (1 - \Delta t)c(t) + \Delta t f(t)$, since $A = B = 1$. If the inputs $f(t)$ can depend on the hidden state $c(t)$ and the discretization step size Δt 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.⁴ 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.

3 HiPPO-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 (Appendix A.1), 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 $[0, t]: \mu^{(t)} = \frac{1}{t} \mathbb{I}_{[0, t]}$. App D, Fig. 5 compares LegS, LegT, and LagT visually, showing the advantages of the scaled measure.

Simply by specifying the desired measure, specializing the HiPPO framework (Sections 2.2, 2.4) yields a new memory mechanism (proof in Appendix D.3).

Theorem 2. *The continuous- (3) and discrete- (4) time dynamics for HiPPO-LegS are:*

$$\begin{aligned} \frac{d}{dt}c(t) &= -\frac{1}{t}Ac(t) + \frac{1}{t}Bf(t) & (3) \\ c_{k+1} &= \left(1 - \frac{A}{k}\right)c_k + \frac{1}{k}Bf_k & (4) \end{aligned} \quad A_{nk} = \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k, \\ 0 & \text{if } n < k \end{cases} \quad B_n = (2n+1)^{\frac{1}{2}}$$

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 E.

³This is known as the Euler method, used for illustration here; our experiments use the more numerically stable Bilinear and ZOH methods. Appendix B.3 provides a self-contained overview of our full discretization framework.

⁴The LSTM cell update is similar, with a parameterization known as ‘‘tied’’ gates [30].

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.

Proposition 3. *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) is invariant to the discretization step size.⁵ By contrast, LegT has a hyperparameter θ for the window size, and both LegT and LagT have a step size hyperparameter Δt in the discrete time case. This hyperparameter is important in practice; Section 2.5 showed that Δt relates to the gates of RNNs, which are known to be sensitive to their parameterization [31, 39, 66]. We empirically demonstrate the benefits of timescale robustness in Section 4.2.

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 $I + \Delta t \cdot A$ and $(I - \Delta t \cdot A)^{-1}$ for arbitrary step sizes Δt . Although this is generically a $O(N^2)$ operation, LegS operators use a fixed A matrix with special structure that turns out to have fast multiplication algorithms for any discretization.⁶

Proposition 4. *Under any generalized bilinear transform discretization (cf. Appendix B.3), each step of the HiPPO-LegS recurrence in equation (4) can be computed in $O(N)$ operations.*

Section 4.3 validates the efficiency of HiPPO layers in practice, where unrolling the discretized versions of Theorem 2 is 10x faster than standard matrix multiplication as done in standard RNNs.

Gradient flow. Much effort has been spent to alleviate the *vanishing gradient problem* in RNNs [56], 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.

Proposition 5. *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 $\left\| \frac{\partial c(t_1)}{\partial f(t_0)} \right\| = \Theta(1/t_1)$.*

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

Proposition 6. *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 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 Empirical Validation

The HiPPO dynamics are simple recurrences that can be easily incorporated into various models. We validate three claims that suggest that when incorporated into a simple RNN, these methods—especially HiPPO-LegS—yield a recurrent architecture with improved memory capability. In Section 4.1, the HiPPO-LegS RNN outperforms other RNN approaches in benchmark long-term dependency tasks for RNNs. Section 4.2 shows that HiPPO-LegS RNN is much more robust to timescale shifts compared to other RNN and neural ODE models. Section 4.3 validates the distinct theoretical advantages of the HiPPO-LegS memory mechanism, allowing fast and accurate online function reconstruction over millions of time steps. Experiment details and additional results are described in Appendix F.

Model Architectures. We first describe briefly how HiPPO memory updates can be incorporated into a simple neural network architecture, yielding a simple RNN model reminiscent of the classic LSTM. Given inputs x_t or features thereof $f_t = u(x_t)$ in any model, the HiPPO framework can be used to memorize the history of features f_t . Thus, given any RNN update function $h_t = \tau(h_{t-1}, x_t)$, we simply replace h_{t-1} with a projected version of the entire history of h , as described in Figure 2. The output of each cell is h_t , which can be passed through any downstream module (e.g. a classification head trained with cross-entropy) to produce predictions. We map the vector h_{t-1} to 1D with a learned encoding before passing to hippo (full architecture in App. F.1).

⁵(4) uses the Euler method for illustration; HiPPO-LegS is invariant to other discretizations (Appendix B.3).

⁶It is known that large families of structured matrices related to orthogonal polynomials are efficient [22].

4.1 Long-range Memory Benchmark Tasks

Models and Baselines. We consider all of the HiPPO methods (**LegT**, **LagT**, and **LegS**). As we show that many different update dynamics seem to lead to LTI systems that give sensible results (Section 2.3), we additionally consider the **Rand** baseline that uses random A and B matrices (normalized appropriately) in its updates, to confirm that the precise derived dynamics are important. LegT additionally considers an additional hyperparameter θ , which should be set to the timescale of the data if known a priori; to show the effect of the timescale, we set it to the ideal value as well as values that are too large and small. The **MGU** is a minimal gated architecture, equivalent to a GRU without the reset gate. The HiPPO architecture we use is simply the MGU with an additional hippo intermediate layer.

We also compare to several RNN baselines designed for long-term dependencies, including the **LSTM** [34], **GRU** [17], **expRNN** [48], and **LMU** [71].⁷

All methods have the same hidden size in our experiments. In particular, for simplicity and to reduce hyperparameters, HiPPO variants tie the memory size N to the hidden state dimension d , so that all methods and baselines have a comparable number of hidden units and parameters. A more detailed comparison of model architectures is in Appendix F.1.

Sequential Image Classification on Permuted MNIST. The permuted MNIST (pMNIST) task feeds inputs to a model pixel-by-pixel in the order of a fixed permutation. The model must process the entire image sequentially – with non-local structure – before outputting a classification label, requiring learning long-term dependencies.

Table 1 shows the validation accuracy on the pMNIST task for the instantiations of our framework and baselines. We highlight that LegS has the best performance of all models. While LegT is close at the optimal hyperparameter θ , its performance can fall off drastically for a mis-specified window length. LagT also performs well at its best hyperparameter Δt .

Table 1 also compares test accuracy of our methods against reported results from the literature, where the LMU was the state-of-the-art for recurrent models. In addition to RNN-based baselines, other sequence models have been evaluated on this dataset, despite being against the spirit of the task because they have global receptive field instead of being strictly sequential. With a test accuracy of 98.3%, HiPPO-LegS sets a true state-of-the-art accuracy on the permuted MNIST dataset.

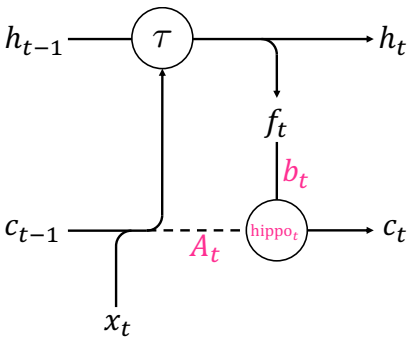


Figure 2: HiPPO incorporated into a simple RNN model. hippo is the HiPPO memory operator which projects the history of the f_t features depending on the chosen measure.

Method	Val. acc. (%)	Model	Test acc.
-LegS	98.34	HiPPO-LegS	98.3
-LagT	98.15	LSTM [31]	95.11
-LegT $\theta=200$	98.0	r-LSTM [69]	95.2
-LegT $\theta=20$	91.75	Dilated RNN [10]	96.1
-Rand	69.93	IndRNN [49]	96.0
LMU	97.08	URLSTM [31]	96.96
ExpRNN	94.67	LMU [71]	97.15
GRU	93.04	Transformer [69]	97.9
MGU	89.37	TCN [5]	97.2
RNN	52.98	TrellisNet [6]	98.13

Table 1: **(Left)** pMNIST validation, average over 3 seeds. Top: Our methods. Bottom: RNN baselines. **(Right)** Reported test accuracies from previous works. Top: Our methods. Middle: Recurrent models. Bottom: Non-recurrent models requiring global receptive field.

Copying task. This standard RNN task [3] directly tests memorization, where models must regurgitate a sequence of tokens seen at the beginning of the sequence. It is well-known that standard models such as LSTMs struggle to solve this task. Appendix F shows the loss for the Copying task with length $L=200$. Our proposed update LegS solves the task almost perfectly, while LegT is very sensitive to the window length hyperparameter. As expected, most baselines make little progress.

⁷In our experiments, LMU refers to the architecture in [71] while LegT uses the one described in Fig. 2.

4.2 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 [39, 66], or the discretization step size Δt of HiPPO-Lag and HiPPO-LegT (Section 2.4). The experiments in Section 4.1 confirm their importance. Fig. 7 (Appendix) and Table 1 ablate these hyperparameters, showing that for example the sliding window length θ must be set correctly for LegT. Additional ablations for other hyperparameters are in Appendix F.

Distribution shift 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 [62, 63]; or a time series may involve the same trajectory evolving at different speeds. Following Kidger et al. [40], we consider the Character Trajectories dataset [4], 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 B.3). Combined with LegS timescale invariance (Prop. 3), 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 [11] which learns a decay between observations, and neural ODE methods which models segments between observations with an ODE.

Table 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 2: Test set accuracy on Character Trajectory classification on out-of-distribution timescales.

Model	LSTM	GRU	GRU-D	ODE-RNN	NCDE	LMU	HiPPO-LegS
100Hz \rightarrow 200Hz	31.9	25.4	23.1	41.8	44.7	6.0	88.8
200Hz \rightarrow 100Hz	28.2	64.6	25.5	31.5	11.3	13.1	90.1
Missing values upsample	24.4	28.2	5.5	4.3	63.9	39.3	94.5
Missing values downsample	34.9	27.3	7.7	7.7	69.7	67.8	94.9

4.3 Theoretical Validation and Scalability

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).

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 2.1. The model only consists of the memory update (Section 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 (Fig. 3). This is a difficult task; the LSTM fails with even sequences of length 1000 (MSE ≈ 0.25). As shown in Table 3, 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. Fig. 3 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 3) and in practice. We implement the fast update in C++ with Pytorch binding and show in Table 3 that it can perform 470,000 time step updates per second on a single CPU core, 10x faster than the LSTM and LMU.⁸

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

Table 3: Function approximation error after 1 million time steps, with 256 hidden units.

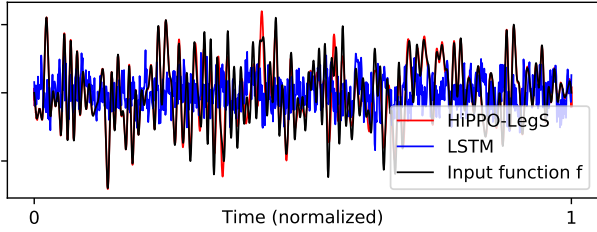


Figure 3: Input function and its reconstructions.

4.4 Additional Experiments

We validate that the HiPPO memory updates also perform well on more generic sequence prediction tasks not exclusively focused on memory. Full results and details for these tasks are in Appendix F.

Sentiment classification task on the IMDB movie review dataset. Our RNNs with HiPPO memory updates perform on par with the LSTM, while other long-range memory approaches such as exprNN perform poorly on this more generic task (Appendix F.6).

Mackey spin glass prediction. This physical simulation task tests the ability to model chaotic dynamical systems. HiPPO-LegS outperforms the LSTM, LMU, and the best hybrid LSTM+LMU model from [71], reducing normalized MSE by 30% (Appendix F.7).

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. We anticipate that the study of this core problem will be useful in improving a variety of sequence models, and are excited about future work on integrating our memory mechanisms with other models in addition to RNNs. We hope to realize the benefits of long-range memory on large-scale tasks such as speech recognition, video processing, and reinforcement learning.

Broader Impact

Our work seeks to understand the foundation of memory in modeling sequential data, which may improve a wide range of applications, each with their own potential benefits and harms. For example, incorporating longer context in language modeling may improve the quality of automated customer services, helpdesks, and personal assistants, but might also facilitate spreading misinformation. Better video processing may produce more coherent video summary for visually impaired users, but might also make automatic surveillance easier.

Our framework presents a principled way to study memory of machine learning models. We speculate that this new representation could be a tool to study how potential biases in training data can get incorporated into the model. This in turn may give us a better handle on how to identify and potentially mitigate the effect of such biases in machine learning models. Though we currently do not have concrete ideas along this line, we encourage future investigation to better understand these learned representations to address fairness issues.

⁸The LMU is only known to be fast with the simple forward Euler discretization [71], but not with more sophisticated methods such as bilinear and ZOH that are required to reduce numerical errors for this task.

Acknowledgments

We thank Avner May, Mayee Chen, Dan Fu, Aditya Grover, and Daniel Lévy for their helpful feedback. We gratefully acknowledge the support of DARPA under Nos. FA87501720095 (D3M), FA86501827865 (SDH), and FA86501827882 (ASED); NIH under No. U54EB020405 (Mobilize), NSF under Nos. CCF1763315 (Beyond Sparsity), CCF1563078 (Volume to Velocity), and 1937301 (RTML); ONR under No. N000141712266 (Unifying Weak Supervision); the Moore Foundation, NXP, Xilinx, LETI-CEA, Intel, IBM, Microsoft, NEC, Toshiba, TSMC, ARM, Hitachi, BASF, Accenture, Ericsson, Qualcomm, Analog Devices, the Okawa Foundation, American Family Insurance, Google Cloud, Stanford HAI AWS cloud credit, Swiss Re, and members of the Stanford DAWN project: Teradata, Facebook, Google, Ant Financial, NEC, VMWare, and Infosys. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views, policies, or endorsements, either expressed or implied, of DARPA, NIH, ONR, or the U.S. Government. Atri Rudra’s research is supported by NSF grant CCF-1763481.

References

- [1] Keivan Alizadeh, Ali Farhadi, and Mohammad Rastegari. Butterfly transform: An efficient FFT based neural architecture design. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.
- [2] George B Arfken and Hans J Weber. *Mathematical methods for physicists*. Elsevier Academic Press, 2005.
- [3] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *The International Conference on Machine Learning (ICML)*, pages 1120–1128, 2016.
- [4] 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. *arXiv preprint arXiv:1811.00075*, 2018.
- [5] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.
- [6] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Trellis networks for sequence modeling. In *The International Conference on Learning Representations (ICLR)*, 2019.
- [7] Raphaël Berthier, Francis Bach, and Pierre Gaillard. Accelerated gossip in networks of given dimension using Jacobi polynomial iterations. *SIAM Journal on Mathematics of Data Science*, 2(1):24–47, 2020.
- [8] John P Boyd. *Chebyshev and Fourier spectral methods*. Courier Corporation, 2001.
- [9] Sarath Chandar, Chinnadhurai Sankar, Eugene Vorontsov, Samira Ebrahimi Kahou, and Yoshua Bengio. Towards non-saturating recurrent units for modelling long-term dependencies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3280–3287, 2019.
- [10] Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael Witbrock, Mark A Hasegawa-Johnson, and Thomas S Huang. Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 77–87, 2017.
- [11] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):1–12, 2018.
- [12] Beijing Chen, Gouenou Coatrieux, Jiasong Wu, Zhifang Dong, Jean Louis Coatrieux, and Huazhong Shu. Fast computation of sliding discrete Tchebichef moments and its application in duplicated regions detection. *IEEE Transactions on Signal Processing*, 63(20):5424–5436, 2015.
- [13] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

- [14] T. S. Chihara. *An introduction to orthogonal polynomials*. Dover Books on Mathematics. Dover Publications, 2011. ISBN 9780486479293.
- [15] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.
- [16] 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.
- [17] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.
- [18] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2019.
- [19] Tri Dao, Christopher M De Sa, and Christopher Ré. Gaussian quadrature for kernel features. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6107–6117, 2017.
- [20] Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Ré. Learning fast algorithms for linear transforms using butterfly factorizations. In *The International Conference on Machine Learning (ICML)*, 2019.
- [21] Tri Dao, Nimit Sohoni, Albert Gu, Matthew Eichhorn, Amit Blonder, Megan Leszczynski, Atri Rudra, and Christopher Ré. Kaleidoscope: An efficient, learnable representation for all structured linear maps. In *The International Conference on Learning Representations (ICLR)*, 2020.
- [22] 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*, pages 1060–1079. SIAM, 2018.
- [23] Raymond A DeCarlo. *Linear systems: A state variable approach with numerical implementation*. Prentice-Hall, Inc., 1989.
- [24] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3844–3852, 2016.
- [25] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL <http://archive.ics.uci.edu/ml>.
- [26] Krzysztof Duda. Accurate, guaranteed stable, sliding discrete Fourier transform [DSP tips & tricks]. *IEEE Signal Processing Magazine*, 27(6):124–127, 2010.
- [27] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural ODEs. In *Advances in Neural Information Processing Systems*, pages 3134–3144, 2019.
- [28] Behrouz Farhang-Boroujeny and Saeed Gazor. Generalized sliding FFT and its application to implementation of block LMS adaptive filters. *IEEE Transactions on Signal Processing*, 42(3):532–538, 1994.
- [29] Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. How to train your neural ODE: the world of Jacobian and kinetic regularization. In *The International Conference on Machine Learning (ICML)*, 2020.
- [30] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.
- [31] 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.

- [32] Caglar Gulcehre, Marcin Moczulski, Misha Denil, and Yoshua Bengio. Noisy activation functions. In *The International Conference on Machine Learning (ICML)*, pages 3059–3068, 2016.
- [33] Mikael Henaff, Arthur Szlam, and Yann LeCun. Recurrent orthogonal networks and long-memory tasks. In *The International Conference on Machine Learning (ICML)*, 2016.
- [34] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [35] Arieh Iserles. *A first course in the numerical analysis of differential equations*. Number 44. Cambridge university press, 2009.
- [36] Eric Jacobsen and Richard Lyons. The sliding DFT. *IEEE Signal Processing Magazine*, 20(2): 74–80, 2003.
- [37] Eric Jacobsen and Richard Lyons. An update to the sliding DFT. *IEEE Signal Processing Magazine*, 21(1):110–111, 2004.
- [38] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004.
- [39] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pages 2342–2350, 2015.
- [40] Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *arXiv preprint arXiv:2005.08926*, 2020.
- [41] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *The International Conference on Learning Representations (ICLR)*, 2015.
- [42] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *The International Conference on Machine Learning (ICML)*, 2020.
- [43] Vitaly Kober. Fast algorithms for the computation of sliding discrete sinusoidal transforms. *IEEE transactions on signal processing*, 52(6):1704–1710, 2004.
- [44] Vitaly Kober. Fast algorithms for the computation of sliding discrete Hartley transforms. *IEEE transactions on signal processing*, 55(6):2937–2944, 2007.
- [45] Thomas William Körner. *Fourier analysis*. Cambridge university press, 1989.
- [46] David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, and Chris Pal. Zoneout: Regularizing RNNs by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.
- [47] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.
- [48] 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.
- [49] 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*, pages 5457–5466, 2018.
- [50] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/P11-1015>.
- [51] Jose A Rosendo Macias and Antonio Gomez Exposito. Efficient computation of the running discrete Haar transform. *IEEE transactions on power delivery*, 21(1):504–505, 2005.

- [52] Michael C Mackey and Leon Glass. Oscillation and chaos in physiological control systems. *Science*, 197(4300):287–289, 1977.
- [53] Stefano Massaroli, Michael Poli, Michelangelo Bin, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Stable neural flows. *arXiv preprint arXiv:2003.08063*, 2020.
- [54] Barzan Mozafari and Mohammad H Savoji. An efficient recursive algorithm and an explicit formula for calculating update vectors of running Walsh-Hadamard transform. In *2007 9th International Symposium on Signal Processing and Its Applications*, pages 1–4. IEEE, 2007.
- [55] Wanli Ouyang and Wai-Kuen Cham. Fast algorithm for Walsh Hadamard transform on sliding windows. *IEEE transactions on pattern analysis and machine intelligence*, 32(1):165–171, 2009.
- [56] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.
- [57] John G Proakis. *Digital signal processing: principles algorithms and applications*. Pearson Education India, 2001.
- [58] Alessio Quaglino, Marco Gallieri, Jonathan Masci, and Jan Koutník. SNODE: Spectral discretization of neural ODEs for system identification. In *The International Conference on Learning Representations (ICLR)*, 2020.
- [59] Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. In *The International Conference on Learning Representations (ICLR)*, 2020.
- [60] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *arXiv preprint arXiv:2003.05997*, 2020.
- [61] 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*, pages 5321–5331, 2019.
- [62] Khaled Saab, Jared Dunnmon, Christopher Ré, Daniel Rubin, and Christopher Lee-Messer. Weak supervision as an efficient approach for automated seizure detection in electroencephalography. *NPJ Digital Medicine*, 3(1):1–12, 2020.
- [63] 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. *Frontiers in neuroinformatics*, 12:83, 2018.
- [64] Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. Adaptive attention span in transformers. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2019.
- [65] G. Szegő. *Orthogonal Polynomials*. Number v.23 in American Mathematical Society colloquium publications. American Mathematical Society, 1967. ISBN 9780821889527.
- [66] Corentin Tallec and Yann Ollivier. Can recurrent neural networks warp time? In *The International Conference on Learning Representations (ICLR)*, 2018.
- [67] Anna Thomas, Albert Gu, Tri Dao, Atri Rudra, and Christopher Ré. Learning compressed transforms with low displacement rank. In *Advances in neural information processing systems*, pages 9052–9060, 2018.
- [68] Lloyd N Trefethen. *Approximation theory and approximation practice*, volume 164. SIAM, 2019.
- [69] 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.
- [70] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

- [71] Aaron Voelker, Ivana Kajić, and Chris Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 15544–15553, 2019.
- [72] Aaron R Voelker and Chris Eliasmith. Improving spiking dynamical networks: Accurate delays, higher-order synapses, and time cells. *Neural computation*, 30(3):569–609, 2018.
- [73] Aaron Russell Voelker. *Dynamical systems in spiking neuromorphic hardware*. PhD thesis, University of Waterloo, 2019.
- [74] 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.
- [75] Jiasong Wu, Lu Wang, Guanyu Yang, Lotfi Senhadji, Limin Luo, and Huazhong Shu. Sliding conjugate symmetric sequency-ordered complex Hadamard transform: fast algorithm and applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 59(6):1321–1334, 2012.
- [76] Greg Yang, Jeffrey Pennington, Vinay Rao, Jascha Sohl-Dickstein, and Samuel S Schoenholz. A mean field theory of batch normalization. In *The International Conference on Learning Representations (ICLR)*, 2019.
- [77] Guofeng Zhang, Tongwen Chen, and Xiang Chen. Performance recovery in digital implementation of analogue systems. *SIAM journal on control and optimization*, 45(6):2207–2223, 2007.
- [78] Han Zhang, Xi Gao, Jacob Unterman, and Tom Arodz. Approximation capabilities of neural ordinary differential equations. In *The International Conference on Machine Learning (ICML)*, 2020.
- [79] 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.

A Related Work

Our work touches on a variety of topics and related work, which we explore in detail.

A.1 Signal Processing and Orthogonal Polynomials

A.1.1 Sliding transforms

The technical contributions in this work build on a rich history of approximation theory in signal processing. Our main framework – orthogonalizing functions with respect to time-varying measures (Section 2) – are related to “online” versions of classical signal processing transforms. In short, these methods compute *specific transforms* on *sliding windows* of *discrete sequences*. Concretely, they calculate $c_{n,k} = \sum_{i=0}^{N-1} f_{k+i} \psi(i,n)$ given signal (f_k) , where $\{\psi(i,n)\}$ is a discrete orthogonal transform. Our technical problem differs in several key aspects:

Specific discrete transforms Examples of sliding transforms considered in the literature include the sliding DFT [26, 28, 36, 37], sliding DCT [43], sliding discrete (Walsh-)Hadamard transform [54, 55, 75], Haar [51], sliding discrete Hartley transform [44], and sliding discrete Chebyshev moments [12]. While each of these address a specific transform, we present a general approach (Section 2) that addresses several transforms at once. Furthermore, we are unaware of sliding transform algorithms for the OPs we consider here, in particular the Legendre and Laguerre polynomials. Our derivations in Appendix D cover Legendre, (generalized) Laguerre, Fourier, and Chebyshev continuous sliding transforms.

Fixed-length sliding windows All mentioned works operate in the sliding window setting, where a fixed-size context window on the discrete signal is taken into account. Our measure-based abstraction for approximation allows considering a new type of *scaled* measure where the window size increases over time, leading to methods with qualitatively different theoretical (Section 3) and empirical properties (Section 4.2). We are not aware of any previous works addressing this scaled setting.

Discrete vs. continuous time Even in the fixed-length sliding window case, our solutions to the “translated measure” problems (e.g., HiPPO-LegT Appendix D.1) solve a continuous-time sliding window problem on an underlying continuous signal, then discretize.

On the other hand, the sliding transform problems calculate transforms directly on a discrete stream. Discrete transforms are equivalent to calculating projection coefficients on a measure (equation (18)) by Gaussian quadrature, which assumes the discrete input is subsampled from a signal at the quadrature nodes [14]. However, since these nodes are non-uniformly spaced in general, the sliding discrete transform is not consistent with a discretization of an underlying continuous signal.

Thus, our main abstraction (Definition 1) has a fundamentally different interpretation than standard transforms, and our approach of first calculating the dynamics of the underlying continuous-time problem (e.g. equation (20)) is correspondingly new.

We remark that our novel *scaled measures* are fundamentally difficult to address with a standard discrete-time based approach. These discrete sliding methods require a fixed-size context in order to have consistent transform sizes, while the scaled measure would require solving transforms with an increasing number of input points over time.

A.1.2 OPs in ML

More broadly, orthogonal polynomials and orthogonal polynomial transforms have recently found applications in various facets of machine learning. For example, Dao et al. [19] leverage the connection between orthogonal polynomials and quadrature to derive rules for computing kernel features in machine learning. More directly, [67] apply parametrized families of structured matrices directly inspired by orthogonal polynomial transforms ([22]) as layers in neural networks. Some particular families of orthogonal polynomials such as the Chebyshev polynomials have desirable approximation properties that find many well-known classical uses in numerical analysis and optimization. More recently, they have been applied to ML models such as graph convolutional neural networks[24], and generalizations such as Gegenbauer and Jacobi polynomials have been used to analyze optimization dynamics[7, 76]. Generalization of orthogonal polynomials and Fourier transform, expressed as products of butterfly

matrices, have found applications in automatic algorithm design [20], model compression [1], and replacing hand-crafted preprocessing in speech recognition [21]. Orthogonal polynomials are known to have various efficiency results [22], and we conjecture that Proposition 4 on the efficiency of HiPPO methods can be extended to arbitrary measures besides the ones considered in this work.

A.2 Memory in Machine Learning

Memory in sequence models Sequential or temporal data in areas such as language, reinforcement learning, and continual learning can involve increasingly long dependencies. However, direct parametric modeling cannot handle inputs of unknown and potentially unbounded lengths. Many modern solutions such as attention [70] and dilated convolutions [5], are functions on finite windows, thus sidestepping the need for an explicit memory representation. While this suffices for certain tasks, these approaches can only process a finite context window instead of an entire sequence. Naively increasing the window length poses significant compute and memory challenges. This has spurred various approaches to extend this fixed context window subjected to compute and storage constraints [6, 15, 18, 42, 59, 60, 64, 74].

We instead focus on the core problem of online processing and memorization of continuous and discrete signals, and anticipate that the study of this foundational problem will be useful in improving a variety of models.

Recurrent memory Recurrent neural networks are a natural tool for modeling sequential data online, with the appealing property of having unbounded context; in other words they can summarize history indefinitely. However, due to difficulties in the optimization process (vanishing/exploding gradients [56]), particular care must be paid to endow them with longer memory. The ubiquitous LSTM [34] and simplifications such as the GRU [17] control the update with gates to smooth the optimization process. With more careful parametrization, the addition of gates alone make RNNs significantly more robust and able to address long-term dependencies [31]. Tallec and Ollivier [66] show that gates are in fact fundamental for recurrent dynamics by allowing time dilations. Many other approaches to endowing RNNs with better memory exist, such as noise injection [32] or non-saturating gates [9], which can suffer from instability issues. A long line of work controls the spectrum of the recurrent updates with (nearly-) orthogonal matrices to control gradients [3], but have been found to be less robust across different tasks [33].

A.3 Directly related methods

LMU The main result of the Legendre Memory Unit [71, 72, 73] is a direct instantiation of our framework using the LegT measure (Section 2.3). The original LMU is motivated by neurobiological advances and approaches the problem from the opposite direction as us: it considers approximating spiking neurons in the frequency domain, while we directly solve an interpretable optimization problem in the time domain. More specifically, they consider time-lagged linear time invariant (LTI) dynamical systems and approximate the dynamics with Padé approximants; Voelker et al. [71] observes that the result also has an interpretation in terms of Legendre polynomials, but not that it is the optimal solution to a natural projection problem. This approach involves heavier machinery, and we were not able to find a complete proof of the update mechanism [71, 72, 73].

In contrast, our approach directly poses the relevant online signal approximation problem, which ties to orthogonal polynomial families and leads to simple derivations of several related memory mechanisms (Appendix D). Our interpretation in time rather than frequency space, and associated derivation (Appendix D.1) for the LegT measure, reveals a different set of approximations stemming from the sliding window, which is confirmed empirically (Appendix F.8).

As the motivations of our work are substantially different from Voelker et al. [71], yet finds the same memory mechanism in a special case, we highlight the potential connection between these sequence models and biological nervous systems as an area of exploration for future work, such as alternative interpretations of our methods in the frequency domain.

We remark that the term LMU in fact refers to a specific recurrent neural network architecture, which interleaves the projection operator with other specific neural network components. By contrast, we use HiPPO to refer to the projection operator in isolation (Theorem 1), which is a function-to-function or sequence-to-sequence operator independent of model. HiPPO is integrated into an RNN architecture

in Section 4, with slight improvements to the LMU architecture, as ablated in Appendices F.2 and F.3. As a standalone module, HiPPO can be used as a layer in other types of models.

Fourier Recurrent Unit The Fourier Recurrent Unit (FRU) [79] uses Fourier basis (cosine and sine) to express the input signal, motivated by the discrete Fourier transform. In particular, each recurrent unit computes the discrete Fourier transform of the input signal for a randomly chosen frequency. It is not clear how discrete transform with respect to other bases (e.g., Legendre, Laguerre, Chebyshev) can in turn yield similar memory mechanisms. We show that FRU is also an instantiation of the HiPPO framework (Appendix D.4), where the Fourier basis can be viewed as orthogonal polynomials z^n on the unit circle $\{z: |z|=1\}$.

Zhang et al. [79] prove that if a timescale hyperparameter is chosen appropriately, FRU has bounded gradients, thus avoiding vanishing and exploding gradients. This essentially follows from the fact that $(1 - \Delta t)^T = \Theta(1)$ if the discretization step size $\Delta t = \Theta(\frac{1}{T})$ is chosen, if the time horizon T is known (cf. Appendices B.3 and E). It is easily shown that this property is not intrinsic to the FRU but to sliding window methods, and is shared by all of our translated measure HiPPO methods (all but HiPPO-LegS in Appendix D). We show the stronger property that HiPPO-LegS, which uses scaling rather than sliding windows, also enjoys bounded gradient guarantees, without needing a well-specified timescale hyperparameter (Proposition 5).

Neural ODEs HiPPO produces linear ODEs that describe the dynamics of the coefficients. Recent work has also incorporated ODEs into machine learning models. Chen et al. [13] introduce neural ODEs, employing general nonlinear ODEs parameterized by neural networks in the context of normalizing flows and time series modeling. Neural ODEs have shown promising results in modeling irregularly sampled time series [40], especially when combined with RNNs [61]. Though neural ODEs are expressive [27, 78], due to their complex parameterization, they often suffer from slow training [29, 53, 58] because of their need for more complicated ODE solvers. On the other hand, HiPPO ODEs are linear and are fast to solve with classical discretization techniques in linear systems, such as Euler method, Bilinear method, and Zero-Order Hold (ZOH) [35].

B Technical Preliminaries

We collect here some technical background that will be used in presenting the general HiPPO framework and in deriving specific HiPPO update rules.

B.1 Orthogonal Polynomials

Orthogonal polynomials are a standard tool for working with function spaces [14, 65]. 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.

B.1.1 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 $\omega^{\text{leg}} = \mathbf{1}_{[-1,1]}$:

$$\frac{2n+1}{2} \int_{-1}^1 P_n(x)P_m(x)dx = \delta_{nm} \tag{5}$$

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 (5) 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 ([2, 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

$$P'_{n+1} = (2n+1)P_n + (2n-3)P_{n-2} + \dots, \quad (6)$$

where the sum stops at P_0 or P_1 .

These equations directly imply

$$P'_n = (2n-1)P_{n-1} + (2n-5)P_{n-3} + \dots \quad (7)$$

and

$$\begin{aligned} (x+1)P'_n(x) &= P'_{n+1} + P'_n - (n+1)P_n \\ &= nP_n + (2n-1)P_{n-1} + (2n-3)P_{n-2} + \dots \end{aligned} \quad (8)$$

These will be used in the derivations of the HiPPO-LegT and HiPPO-LegS updates, respectively.

B.1.2 Properties of Laguerre Polynomials

The standard Laguerre polynomials $L_n(x)$ are defined to be orthogonal with respect to the weight function e^{-x} supported on $[0, \infty)$, while the generalized Laguerre polynomials (also called associated Laguerre polynomials) $L_n^{(\alpha)}$ are defined to be orthogonal with respect to the weight function $x^\alpha e^{-x}$ also supported on $[0, \infty)$:

$$\int_0^\infty x^\alpha e^{-x} L_n^{(\alpha)}(x) L_m^{(\alpha)}(x) dx = \frac{(n+\alpha)!}{n!} \delta_{n,m}. \quad (9)$$

Also, they satisfy

$$L_n^{(\alpha)}(0) = \binom{n+\alpha}{n} = \frac{\Gamma(n+\alpha+1)}{\Gamma(n+1)\Gamma(\alpha+1)}. \quad (10)$$

The standard Laguerre polynomials correspond to the case of $\alpha = 0$ of generalized Laguerre polynomials.

Derivatives of generalized Laguerre polynomials We note the following recurrence relations on generalized Laguerre polynomials ([2, Chapter 13.2]):

$$\begin{aligned}\frac{d}{dx}L_n^{(\alpha)}(x) &= -L_{n-1}^{(\alpha+1)}(x) \\ L_n^{(\alpha+1)}(x) &= \sum_{i=0}^n L_i^{(\alpha)}(x).\end{aligned}$$

These equations imply

$$\frac{d}{dt}L_n^{(\alpha)}(x) = -L_0^{(\alpha)}(x) - L_1^{(\alpha)}(x) - \dots - L_{n-1}^{(\alpha)}(x).$$

B.1.3 Properties of Chebyshev polynomials

Let T_n be the classical Chebyshev polynomials (of the first kind), defined to be orthogonal with respect to the weight function $(1-x^2)^{1/2}$ supported on $[-1, 1]$, and let p_n be the normalized version of T_n (i.e. with norm 1):

$$\begin{aligned}\omega^{\text{cheb}} &= (1-x^2)^{-1/2}\mathbb{I}_{(-1,1)}, \\ p_n(x) &= \sqrt{\frac{2}{\pi}}T_n(x) \quad \text{for } n \geq 1, \\ p_0(x) &= \frac{1}{\sqrt{\pi}}.\end{aligned}$$

Note that ω^{cheb} is not normalized (it integrates to π).

Derivatives of Chebyshev polynomials The chebyshev polynomials satisfy

$$2T_n(x) = \frac{1}{n+1} \frac{d}{dx}T_{n+1}(x) - \frac{1}{n-1} \frac{d}{dx}T_{n-1}(x) \quad n=2,3,\dots$$

By telescoping this series, we obtain

$$\frac{1}{n}T_n' = \begin{cases} 2(T_{n-1} + T_{n-3} + \dots + T_2) + T_0 & n \text{ odd} \\ 2(T_{n-1} + T_{n-3} + \dots + T_1) & n \text{ even} \end{cases}. \quad (11)$$

Translated Chebyshev polynomials We will also consider shifting and scaling the Chebyshev polynomials to be orthogonal on the interval $[t-\theta, t]$ for fixed length θ .

The normalized (probability) measure is

$$\omega(t,x) = \frac{2}{\theta\pi} \omega^{\text{cheb}}\left(\frac{2(x-t)}{\theta} + 1\right) = \frac{1}{\theta\pi} \left(\frac{x-t}{\theta} + 1\right)^{-1/2} \left(-\frac{x-t}{\theta}\right)^{-1/2} \mathbb{I}_{(t-\theta,t)}.$$

The orthonormal polynomial basis is

$$p_n(t,x) = \sqrt{\pi} p_n\left(\frac{2(x-t)}{\theta} + 1\right).$$

In terms of the original Chebyshev polynomials, these are

$$\begin{aligned}p_n(t,x) &= \sqrt{2}T_n\left(\frac{2(x-t)}{\theta} + 1\right) \quad \text{for } n \geq 1, \\ p_0^{(t)} &= T_0\left(\frac{2(x-t)}{\theta} + 1\right).\end{aligned}$$

B.2 Leibniz Integral Rule

As part of our standard strategy for deriving HiPPO update rules (Appendix C), 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) 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:

$$\begin{aligned} \frac{\partial}{\partial t} \int_{\alpha(t)}^{\beta(t)} f(x,t) dx &= \frac{\partial}{\partial t} \int f(x,t) \mathbb{I}_{[\alpha(t), \beta(t)]}(x) dx \\ &= \int \frac{\partial}{\partial t} f(x,t) \mathbb{I}_{[\alpha(t), \beta(t)]}(x) dx + \int f(x,t) \frac{\partial}{\partial t} \mathbb{I}_{[\alpha(t), \beta(t)]}(x) dx \\ &= \int \frac{\partial}{\partial t} f(x,t) \mathbb{I}_{[\alpha(t), \beta(t)]}(x) dx + \int f(x,t) (\beta'(t) \delta_{\beta(t)}(x) - \alpha'(t) \delta_{\alpha(t)}(x)) dx \\ &= \int_{\alpha(t)}^{\beta(t)} \frac{\partial}{\partial t} f(x,t) dx - \alpha'(t) f(\alpha(t), t) + \beta'(t) f(\beta(t), t). \end{aligned}$$

B.3 ODE Discretization

In our framework, time series inputs will be modeled with a continuous function and then discretized. Here we provide some background on ODE discretization methods, including a new discretization that applies to a specific type of ODE that our new method encounters.

The general formulation of an ODE is $\frac{d}{dt} c(t) = f(t, c(t))$. We will also focus on the linear time-invariant ODE of the form $\frac{d}{dt} c(t) = Ac(t) + Bf(t)$ for some input function $f(t)$, as a special case. The general methodology for discretizing the ODE, for step size Δt , is to rewrite the ODE as

$$c(t + \Delta t) - c(t) = \int_t^{t + \Delta t} f(s, c(s)) ds, \quad (12)$$

then approximate the RHS integral.

Many ODE discretization methods corresponds to different ways to approximate the RHS integral:

Euler (aka forward Euler). To approximate the RHS of equation (12), keep the left endpoint $\Delta t f(t, c(t))$. For the linear ODE, we get:

$$c(t + \Delta t) = (I + \Delta t A) c(t) + \Delta t B f(t).$$

Backward Euler. To approximate the RHS of equation (12), keep the right endpoint $\Delta t f(t + \Delta t, c(t + \Delta t))$. For the linear ODE, we get the linear equation and the update:

$$\begin{aligned} c(t + \Delta t) - \Delta t A c(t + \Delta t) &= c(t) + \Delta t B f(t) \\ c(t + \Delta t) &= (I - \Delta t A)^{-1} c(t) + \Delta t (I - \Delta t A)^{-1} B f(t). \end{aligned}$$

Bilinear (aka Trapezoid rule, aka Tustin's method). To approximate the RHS of equation (12), average the endpoints $\Delta t \frac{f(t, c(t)) + f(t + \Delta t, c(t + \Delta t))}{2}$. For the linear ODE, again we get a linear equation and the update:

$$\begin{aligned} c(t + \Delta t) - \frac{\Delta t}{2} A c(t + \Delta t) &= (I + \Delta t / 2 A) c(t) + \Delta t B f(t) \\ c(t + \Delta t) &= (I - \Delta t / 2 A)^{-1} (I + \Delta t / 2 A) c(t) + \Delta t (I - \Delta t / 2 A)^{-1} B f(t). \end{aligned}$$

Generalized Bilinear Transformation (GBT). This method [77] approximates the RHS of equation (12) by taking a weighted average of the endpoints $\Delta t[(1-\alpha)f(t,c(t)) + \alpha f(t+\Delta t,c(t+\Delta t))]$, for some parameter $\alpha \in [0,1]$. For the linear ODE, again we get a linear equation and the update:

$$\begin{aligned} c(t+\Delta t) - \Delta t \alpha A c(t+\Delta t) &= (I + \Delta t(1-\alpha)A)c(t) + \Delta t B f(t) \\ c(t+\Delta t) &= (I - \Delta t \alpha A)^{-1} (I + \Delta t(1-\alpha)A)c(t) + \Delta t (I - \Delta t \alpha A)^{-1} B f(t). \end{aligned} \quad (13)$$

GBT generalizes the three methods mentioned above: forward Euler corresponds to $\alpha=0$, backward Euler to $\alpha=1$, and bilinear to $\alpha=1/2$.

We also note another method called *Zero-order Hold (ZOH)* [23] that specializes to linear ODEs. The RHS of equation (12) is calculated in closed-form assuming constant input f between t and $t+\Delta t$. This yields the update $c(t+\Delta t) = e^{\Delta t A} c(t) + \left(\int_{\tau=0}^{\Delta t} e^{\tau A} d\tau\right) B f(t)$. If A is invertible, this can be simplified as $c(t+\Delta t) = e^{\Delta t A} c(t) + A^{-1}(e^{\Delta t A} - I) B f(t)$.

HiPPO-LegS invariance to discretization step size. In the case of HiPPO-LegS, we have a linear ODE of the form $\frac{d}{dt}c(t) = \frac{1}{t}Ac(t) + \frac{1}{t}Bf(t)$. Adapting the GBT discretization (which generalizes forward/backward Euler and bilinear) to this linear ODE, we obtain:

$$\begin{aligned} c(t+\Delta t) - \Delta t \alpha \frac{1}{t+\Delta t} A c(t+\Delta t) &= \left(I + \Delta t(1-\alpha) \frac{1}{t} A\right) c(t) + \Delta t \frac{1}{t} B f(t) \\ c(t+\Delta t) &= \left(I - \frac{\Delta t}{t+\Delta t} \alpha A\right)^{-1} \left(I + \frac{\Delta t}{t} (1-\alpha) A\right) c(t) + \frac{\Delta t}{t} \left(I - \frac{\Delta t}{t+\Delta t} \alpha A\right)^{-1} B f(t). \end{aligned}$$

We highlight that this system is invariant to the discretization step size Δt . Indeed, if $c^{(k)} := c(k\Delta t)$ and $f_k := f(k\Delta t)$ then we have the recurrence

$$c^{(k+1)} = \left(I - \frac{1}{k+1} \alpha A\right)^{-1} \left(I + \frac{1}{k} (1-\alpha) A\right) c^{(k)} + \frac{1}{k} \left(I - \frac{1}{k+1} \alpha A\right)^{-1} B f_k,$$

which does not depend on Δt .

Ablation: comparison between different discretization methods To understand the impact of approximation error in discretization, in Fig. 4, we show the absolute error for the HiPPO-LegS updates in function approximation (Appendix F.8) 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.

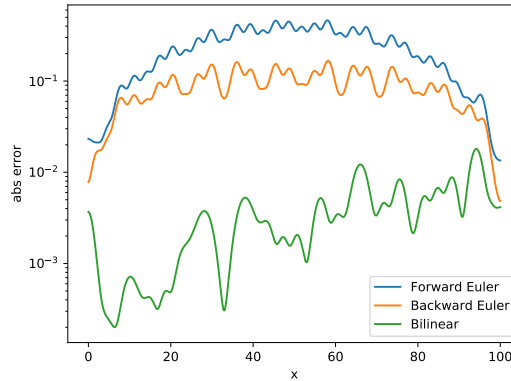


Figure 4: Absolute error for different discretization methods. Forward and backward Euler are generally not very accurate, while bilinear yields more accurate approximation.

C General HiPPO Framework

We present the general HiPPO framework, as described in Section 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

$$g^{(t)} := \operatorname{argmin}_{g \in \mathcal{G}} \|f_{\leq t} - g\|_{\mu^{(t)}}, \quad \text{and} \quad g^{(t)} = \sum_{n=0}^{N-1} c_n(t) g_n^{(t)}.$$

The first step refers to the proj_t operator and the second the coef_t operator in Definition 1.

We focus on the case where the coefficients $c(t)$ has the form of a linear ODE satisfying $\frac{d}{dt}c(t) = A(t)c(t) + B(t)f(t)$ for some $A(t) \in \mathbb{R}^{N \times N}$, $B(t) \in \mathbb{R}^{N \times 1}$.

We first describe the parameters of the hippo operator (a measure and basis) in more detail in Appendix C.1. We define the projection proj_t and coefficient coef_t operators in Appendix C.2. Then we give a general strategy to calculate these coefficients $c(t)$, by deriving a differential equation that governs the coefficient dynamics (Appendix C.3). Finally we discuss how to turn the continuous hippo operator into a discrete one that can be applied to sequence data (Appendix C.4).

C.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 $\omega(t, x) := \frac{d\mu^{(t)}}{dx}(x)$ 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), and define

$$p_n(t, x) = p_n^{(t)}(x). \quad (14)$$

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

$$\chi(t, x) = \chi^{(t)}(x), \quad (15)$$

the functions $p_n(x)\chi(x)$ 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

$$\zeta(t) = \int \frac{\omega}{\chi^2} = \int \frac{\omega^{(t)}(x)}{(\chi^{(t)}(x))^2} dx \quad (16)$$

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 $\nu^{(t)}$

$$g_n^{(t)} = \lambda_n \zeta(t)^{\frac{1}{2}} p_n^{(t)} \chi^{(t)}, \quad n \in \mathbb{N}. \quad (17)$$

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.

We will only use this tilting in the case of Laguerre (Appendix D.2 and Chebyshev (Appendix D.5).

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).

C.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 $f(x)_{x \leq t} = f(x)_{x \leq t}$ 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

$$\begin{aligned} c_n(t) &= \langle f_{\leq t}, g_n^{(t)} \rangle_{\nu^{(t)}} \\ &= \int f g_n^{(t)} \frac{\omega^{(t)}}{\zeta(t)(\chi^{(t)})^2} \\ &= \zeta(t)^{-\frac{1}{2}} \lambda_n \int f p_n^{(t)} \frac{\omega^{(t)}}{\chi^{(t)}}. \end{aligned} \quad (18)$$

Reconstruction At any time t , $f_{\leq t}$ can be explicitly reconstructed as

$$\begin{aligned} f_{\leq t} \approx g^{(t)} &:= \sum_{n=0}^{N-1} \langle f_{\leq t}, g_n^{(t)} \rangle_{\nu^{(t)}} \frac{g_n^{(t)}}{\|g_n^{(t)}\|_{\nu^{(t)}}^2} \\ &= \sum_{n=0}^{N-1} \lambda_n^{-2} c_n(t) g_n^{(t)} \\ &= \sum_{n=0}^{N-1} \lambda_n^{-1} \zeta^{\frac{1}{2}} c_n(t) p_n^{(t)} \chi^{(t)}. \end{aligned} \quad (19)$$

Equation (19) 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 $c(t) = (c_n(t))_{n \in [N]}$.

C.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 (18), 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 (18),

$$\begin{aligned} \frac{d}{dt}c_n(t) &= \zeta(t)^{-\frac{1}{2}}\lambda_n \int f(x) \left(\frac{\partial}{\partial t} p_n(t,x) \right) \frac{\omega}{\chi}(t,x) dx \\ &\quad + \int f(x) \left(\zeta^{-\frac{1}{2}}\lambda_n p_n(t,x) \right) \left(\frac{\partial}{\partial t} \frac{\omega}{\chi}(t,x) \right) dx. \end{aligned} \quad (20)$$

Here we have made use of the assumption that ζ is constant for all t .

Let $c(t) \in \mathbb{R}^{N-1}$ 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 Eq. (20) 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 Eq. (20) 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 [71] used a different scaling. Appendix D.1 will recover the LMU by choosing different λ_n for the LegT measure.

Suppose that equation (20) reduced to dynamics of the form

$$\frac{d}{dt}c(t) = -A(t)c(t) + B(t)f(t).$$

Then, letting $\Lambda = \text{diag}_{n \in [N]} \{\lambda_n\}$,

$$\frac{d}{dt}\Lambda^{-1}c(t) = -\Lambda^{-1}A(t)\Lambda^{-1}c(t) + \Lambda^{-1}B(t)f(t).$$

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

$$\frac{d}{dt}c(t) = -(\Lambda^{-1}A(t)\Lambda)c(t) + (\Lambda^{-1}B(t))f(t) \quad (21)$$

$$f_{\leq t} \approx g^{(t)} = \sum_{n=0}^{N-1} \zeta^{\frac{1}{2}} c_n(t) p_n^{(t)} \chi^{(t)} \quad (22)$$

These are the hippo and proj_t operators.

C.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 B.3) can be applied

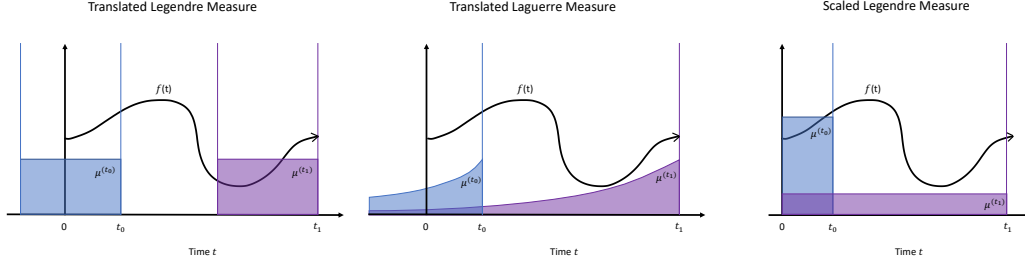


Figure 5: **Illustration of HiPPO measures.** At time t_0 , the history of a function $f(x)_{x \leq t_0}$ is summarized by polynomial approximation with respect to the measure $\mu^{(t_0)}$ (blue), and similarly for time t_1 (purple). (Left) The Translated Legendre measure (**LegT**) assigns weight in the window $[t-\theta, t]$. For small t , $\mu^{(t)}$ is supported on a region $x < 0$ where f is not defined. When t is large, the measure is not supported near 0, causing the projection of f to forget the beginning of the function. (Middle) The Translated Laguerre (**LagT**) measure decays the past exponentially. It does not forget, but also assigns weight on $x < 0$. (Right) The Scaled Legendre measure (**LegS**) weights the entire history $[0, t]$ uniformly.

to turn this into discrete memory updates. Thus we overload these operators, i.e. hippo either defines an ODE of the form

$$\frac{d}{dt}c(t) = A(t)c(t) + B(t)f(t)$$

or a recurrence

$$c_t = A_t c_{t-1} + B_t f_t,$$

whichever is clear from context.

Appendix F.5 validates the framework by applying (20) and (19) to approximate a synthetic function.

D Derivations of HiPPO Projection Operators

We derive the memory updates associated with the translated Legendre (LegT) and translated Laguerre (LagT) measures as presented in Section 2.3, along with the scaling Legendre (LegS) measure (Section 3). To show the generality of the framework, we also derive memory updates with Fourier basis (recovering the Fourier Recurrent Unit [79]) and with Chebyshev basis.

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 C. In particular, the definition of the coefficients (18) and reconstruction (19) does not change, and we only consider how to calculate the coefficients dynamics (20).

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 (equation (20)) 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 derivations in Appendices D.1 and D.2 prove Theorem 1, and the derivations in Appendix D.3 prove Theorem 2. Appendices D.4 and D.5 show additional results for Fourier-based bases.

Figure 5 illustrates the overall framework when we use Legendre and Laguerre polynomials as the basis, contrasting our main families of time-varying measures $\mu^{(t)}$.

D.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:

$$\begin{aligned}\omega(t, x) &= \frac{1}{\theta} \mathbb{1}_{[t-\theta, t]} \\ p_n(t, x) &= (2n+1)^{1/2} P_n\left(\frac{2(x-t)}{\theta} + 1\right) \\ g_n(t, x) &= \lambda_n p_n(t, x).\end{aligned}$$

Here, we have used no tilting so $\chi = 1$ and $\zeta = 1$ (equations (15) and (16)). 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 B.1.1).

$$\begin{aligned}\frac{\partial}{\partial t} g_n(t, x) &= \lambda_n (2n+1)^{\frac{1}{2}} \cdot \frac{-2}{\theta} P_n'\left(\frac{2(x-t)}{\theta} + 1\right) \\ &= \lambda_n (2n+1)^{\frac{1}{2}} \frac{-2}{\theta} \left[(2n-1) P_{n-1}\left(\frac{2(x-t)}{\theta} + 1\right) + (2n-5) P_{n-3}\left(\frac{2(x-t)}{\theta} + 1\right) + \dots \right] \\ &= -\lambda_n (2n+1)^{\frac{1}{2}} \frac{2}{\theta} \left[\lambda_{n-1}^{-1} (2n-1)^{\frac{1}{2}} g_{n-1}(t, x) + \lambda_{n-3}^{-1} (2n-3)^{\frac{1}{2}} g_{n-3}(t, x) + \dots \right].\end{aligned}$$

We have used equation (7) 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 (19), 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 (20) 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

$$\begin{aligned}
\frac{d}{dt}c(t) &= -\frac{1}{\theta} Ac(t) + \frac{1}{\theta} Bf(t) \\
A_{nk} &= (2n+1)^{\frac{1}{2}} (2k+1)^{\frac{1}{2}} \begin{cases} 1 & \text{if } k \leq n \\ (-1)^{n-k} & \text{if } k \geq n \end{cases} \\
B_n &= (2n+1)^{\frac{1}{2}}.
\end{aligned}$$

The second case takes $\lambda_n = (2n+1)^{\frac{1}{2}} (-1)^n$. This yields

$$\begin{aligned}
\frac{d}{dt}c(t) &= -\frac{1}{\theta} Ac(t) + \frac{1}{\theta} Bf(t) \\
A_{nk} &= (2n+1) \begin{cases} (-1)^{n-k} & \text{if } k \leq n \\ 1 & \text{if } k \geq n \end{cases} \\
B_n &= (2n+1) (-1)^n.
\end{aligned}$$

This is exactly the LMU update equation.

Reconstruction By equation (19), 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 Derivation for Translated Laguerre (HiPPO-LagT)

We consider measures based on the generalized *Laguerre* polynomials. For a fixed $\alpha \in \mathbb{R}$, these polynomials $L^{(\alpha)}(t-x)$ are orthogonal with respect to the measure $x^\alpha e^{-x}$ on $[0, \infty)$ (cf. Appendix B.1.2). This derivation will involve tilting the measure governed by another parameter β .

The result in Theorem 1 for HiPPO-LagT is for the case $\alpha = 0, \beta = 1$, corresponding to the basic Laguerre polynomials and no tilting.

Measure and Basis We flip and translate the generalized Laguerre weight function and polynomials from $[0, \infty)$ to $(-\infty, t]$. The normalization is found using equation (9).

$$\begin{aligned}\omega(t, x) &= \begin{cases} (t-x)^\alpha e^{x-t} & \text{if } x \leq t \\ 0 & \text{if } x > t \end{cases} \\ &= (t-x)^\alpha e^{-(t-x)} \mathbb{I}_{(-\infty, t]} \\ p_n(t, x) &= \frac{\Gamma(n+1)^{\frac{1}{2}}}{\Gamma(n+\alpha+1)^{\frac{1}{2}}} L_n^{(\alpha)}(t-x)\end{aligned}$$

Tilted Measure We choose the following tilting χ

$$\chi(t, x) = (t-x)^\alpha \exp\left(-\frac{1-\beta}{2}(t-x)\right) \mathbb{I}_{(-\infty, t]}$$

for some fixed $\beta \in \mathbb{R}$. The normalization is (constant across all t)

$$\begin{aligned}\zeta &= \int \frac{\omega}{\chi^2} = \int (t-x)^{-\alpha} e^{-\beta(t-x)} \mathbb{I}_{(-\infty, t]} dx \\ &= \Gamma(1-\alpha) \beta^{\alpha-1},\end{aligned}$$

so the tilted measure has density

$$\zeta(t)^{-1} \frac{\omega(t)}{(\chi(t))^2} = \Gamma(1-\alpha)^{-1} \beta^{1-\alpha} (t-x)^{-\alpha} \exp(-\beta(t-x)) \mathbb{I}_{(-\infty, t]}.$$

We choose

$$\lambda_n = \frac{\Gamma(n+\alpha+1)^{\frac{1}{2}}}{\Gamma(n+1)^{\frac{1}{2}}}$$

to be the norm of the generalized Laguerre polynomial $L_n^{(\alpha)}$, so that $\lambda_n p_n^{(t)} = L_n^{(\alpha)}(t-x)$, and (following equation (17)) the basis for $\nu^{(t)}$ is

$$\begin{aligned}g_n^{(t)} &= \lambda_n \zeta^{\frac{1}{2}} p_n^{(t)} \chi^{(t)} \\ &= \zeta^{\frac{1}{2}} \chi^{(t)} L_n^{(\alpha)}(t-x)\end{aligned}\tag{23}$$

Derivatives We first calculate the density ratio

$$\frac{\omega}{\chi}(t, x) = \exp\left(-\frac{1+\beta}{2}(t-x)\right) \mathbb{I}_{(-\infty, t]}.$$

and its derivative

$$\frac{\partial}{\partial t} \frac{\omega}{\chi}(t, x) = -\left(\frac{1+\beta}{2}\right) \frac{\omega}{\chi}(t, x) + \exp\left(-\left(\frac{1+\beta}{2}\right)(t-x)\right) \delta_t.$$

The derivative of Laguerre polynomials can be expressed as linear combinations of other Laguerre polynomials (cf. Appendix B.1.2).

$$\begin{aligned}\frac{\partial}{\partial t} \lambda_n p_n(t, x) &= \frac{\partial}{\partial t} L_n^{(\alpha)}(t-x) \\ &= -L_0^{(\alpha)}(t-x) - \dots - L_{n-1}^{(\alpha)}(t-x) \\ &= -\lambda_0 p_0(t, x) - \dots - \lambda_{n-1} p_{n-1}(t, x)\end{aligned}$$

Coefficient Dynamics Plugging these derivatives into equation (20) (obtained from differentiating the coefficient equation (18)), where we suppress the dependence on x for convenience:

$$\begin{aligned}
\frac{d}{dt}c_n(t) &= \zeta^{-\frac{1}{2}} \int f \cdot \left(\frac{\partial}{\partial t} \lambda_n p_n^{(t)} \right) \frac{\omega^{(t)}}{\chi^{(t)}} \\
&\quad + \int f \cdot \left(\zeta^{-\frac{1}{2}} \lambda_n p_n^{(t)} \right) \left(\frac{\partial}{\partial t} \frac{\omega^{(t)}}{\chi^{(t)}} \right) \\
&= - \sum_{k=0}^{n-1} \int f \cdot \left(\zeta^{-\frac{1}{2}} \lambda_k p_k^{(t)} \chi^{(t)} \right) \frac{\omega^{(t)}}{(\chi^{(t)})^2} \\
&\quad - \left(\frac{1+\beta}{2} \right) \int f \cdot \left(\zeta^{-\frac{1}{2}} \lambda_n p_n^{(t)} \right) \frac{\omega^{(t)}}{\chi^{(t)}} + f(t) \cdot \zeta^{-\frac{1}{2}} L_n^{(\alpha)}(0) \\
&= - \sum_{k=0}^{n-1} c_k(t) - \left(\frac{1+\beta}{2} \right) c_n(t) + \Gamma(1-\alpha)^{-\frac{1}{2}} \beta^{\frac{1-\alpha}{2}} \binom{n+\alpha}{n} f(t).
\end{aligned}$$

We then get

$$\begin{aligned}
\frac{d}{dt}c(t) &= -Ac(t) + Bf(t) \\
A &= \begin{bmatrix} \frac{1+\beta}{2} & 0 & \dots & 0 \\ 1 & \frac{1+\beta}{2} & \dots & 0 \\ \vdots & & \ddots & \\ 1 & 1 & \dots & \frac{1+\beta}{2} \end{bmatrix} \\
B &= \zeta^{-\frac{1}{2}} \cdot \begin{bmatrix} \binom{\alpha}{0} \\ \vdots \\ \binom{N-1+\alpha}{N-1} \end{bmatrix}
\end{aligned} \tag{24}$$

Reconstruction By equation (19), at every time t , for $x \leq t$,

$$\begin{aligned}
f(x) \approx g^{(t)}(x) &= \sum_{n=0}^{N-1} \lambda_n^{-1} \zeta^{\frac{1}{2}} c_n(t) p_n^{(t)} \chi^{(t)} \\
&= \sum_n \frac{n!}{(n+\alpha)!} \zeta^{\frac{1}{2}} c_n(t) \cdot L_n^{(\alpha)}(t-x) \cdot (t-x)^\alpha e^{(\frac{\beta-1}{2})(t-x)}.
\end{aligned}$$

Normalized Dynamics Finally, following equations (21) and (22) to convert these to dynamics on the orthonormal basis of the normalized (probability) measure $\nu^{(t)}$ leads to the following hippo operator

$$\begin{aligned}
\frac{d}{dt}c(t) &= -Ac(t) + Bf(t) \\
A &= -\Lambda^{-1} \begin{bmatrix} \frac{1+\beta}{2} & 0 & \dots & 0 \\ 1 & \frac{1+\beta}{2} & \dots & 0 \\ \vdots & & \ddots & \\ 1 & 1 & \dots & \frac{1+\beta}{2} \end{bmatrix} \Lambda \\
B &= \Gamma(1-\alpha)^{-\frac{1}{2}} \beta^{\frac{1-\alpha}{2}} \cdot \Lambda^{-1} \begin{bmatrix} \binom{\alpha}{0} \\ \vdots \\ \binom{N-1+\alpha}{N-1} \end{bmatrix} \\
\Lambda &= \text{diag} \left\{ \frac{\Gamma(n+\alpha+1)^{\frac{1}{2}}}{\Gamma(n+1)^{\frac{1}{2}}} \right\}_{n \in [N]}
\end{aligned} \tag{25}$$

and correspondingly a proj_t operator:

$$f(x) \approx g^{(t)}(x) = \Gamma(1-\alpha)^{\frac{1}{2}} \beta^{-\frac{1-\alpha}{2}} \sum_n c_n(t) \cdot \frac{\Gamma(n+1)^{\frac{1}{2}}}{\Gamma(n+\alpha+1)^{\frac{1}{2}}} \cdot L_n^{(\alpha)}(t-x) \cdot (t-x)^\alpha e^{(\frac{\beta-1}{2})(t-x)}. \tag{26}$$

D.3 Derivation for Scaled Legendre (HiPPO-LegS)

As discussed in Section 3, the scaled Legendre is our only method that uses a measure with varying width.

Measure and Basis We instantiate the framework in the case

$$\omega(t, x) = \frac{1}{t} \mathbb{I}_{[0, t]} \quad (27)$$

$$g_n(t, x) = p_n(t, x) = (2n+1)^{\frac{1}{2}} P_n\left(\frac{2x}{t} - 1\right) \quad (28)$$

Here, P_n are the basic Legendre polynomials (Appendix B.1.1). 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 (8)).

$$\begin{aligned} \frac{\partial}{\partial t} g_n(t, x) &= -(2n+1)^{\frac{1}{2}} t^{-1} (z+1) P_n'(z) \\ &= -(2n+1)^{\frac{1}{2}} t^{-1} [nP_n(z) + (2n-1)P_{n-1}(z) + (2n-3)P_{n-2}(z) + \dots] \\ &= -t^{-1} (2n+1)^{\frac{1}{2}} \left[n(2n+1)^{-\frac{1}{2}} g_n(t, x) + (2n-1)^{\frac{1}{2}} g_{n-1}(t, x) + (2n-3)^{\frac{1}{2}} g_{n-2}(t, x) + \dots \right] \end{aligned}$$

Coefficient Dynamics Plugging these into (20), 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 (3):

$$\begin{aligned} \frac{d}{dt} c(t) &= -\frac{1}{t} A c(t) + \frac{1}{t} B f(t) \quad (29) \\ A_{nk} &= \begin{cases} (2n+1)^{1/2} (2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k, \\ 0 & \text{if } n < k \end{cases} \\ B_n &= (2n+1)^{\frac{1}{2}} \end{aligned}$$

Alternatively, we can write this as

$$\frac{d}{dt} c(t) = -t^{-1} D [M D^{-1} c(t) + \mathbf{1} f(t)], \quad (30)$$

where $D := \text{diag} \left[(2n+1)^{\frac{1}{2}} \right]_{n=0}^{N-1}$, $\mathbf{1}$ is the all ones vector, and the state matrix M is

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 1 & 2 & 0 & 0 & \dots & 0 \\ 1 & 3 & 3 & 0 & \dots & 0 \\ 1 & 3 & 5 & 4 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 3 & 5 & 7 & \dots & N \end{bmatrix}, \quad \text{that is,} \quad M_{nk} = \begin{cases} 2k+1 & \text{if } k < n \\ k+1 & \text{if } k = n \\ 0 & \text{if } k > n \end{cases}$$

Equation (29) is a linear dynamical system, except dilated by a time-varying factor t^{-1} , which arises from the scaled measure.

Reconstruction By equation (19), 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.4 Derivation for Fourier Bases

In the remainder of Appendix D, we consider some additional bases which are analyzable under the HiPPO framework. These use measures and bases related to various forms of the Fourier transform.

D.4.1 Translated Fourier

Similar to the LMU, the sliding Fourier measure also has a fixed window length θ parameter and slides it across time.

Measure The Fourier basis $e^{2\pi i n x}$ (for $n = 0, \dots, N-1$) can be seen as an orthogonal polynomials basis z^n with respect to the uniform measure on the unit circle $\{z: |z|=1\}$. By a change of variable $z \rightarrow e^{2\pi i x}$ (and thus changing the domain from the unit circle to $[0, 1]$), we obtain the usual Fourier basis $e^{2\pi i n x}$. The complex inner product $\langle f, g \rangle$ is defined as $\int_0^1 f(x) \overline{g(x)} dx$. Note that the basis $e^{2\pi i n x}$ is orthonormal.

For each t , we will use a sliding measure uniform on $[t-\theta, t]$ and rescale the basis as $e^{2\pi i n \frac{t-x}{\theta}}$ (so they are still orthonormal, i.e., have norm 1):

$$\begin{aligned} \omega(t, x) &= \frac{1}{\theta} \mathbb{1}_{[t-\theta, t]} \\ p_n(t, x) &= e^{2\pi i n \frac{t-x}{\theta}}. \end{aligned}$$

We use no tilting (i.e., $\chi(t, x) = 1$).

Derivatives

$$\begin{aligned} \frac{\partial}{\partial t} \omega(t, x) &= \frac{1}{\theta} \delta_t - \frac{1}{\theta} \delta_{t-\theta} \\ \frac{\partial}{\partial t} p_n(t, x) &= \frac{2\pi i n}{\theta} e^{2\pi i n \frac{t-x}{\theta}} = \frac{2\pi i n}{\theta} p_n(t, x). \end{aligned}$$

Coefficient Updates Plugging into equation (20) yields

$$\begin{aligned} \frac{d}{dt} c_n(t) &= \frac{2\pi i n}{\theta} c_n(t) + \frac{1}{\theta} f(t) p_n(t, t) - \frac{1}{\theta} f(t-\theta) p_n(t, t-\theta) \\ &= \frac{2\pi i n}{\theta} c_n(t) + \frac{1}{\theta} f(t) - \frac{1}{\theta} f(t-\theta). \end{aligned}$$

Note that $p_n(t, t) = p_n(t, t - \theta) = 1$. Additionally, we no longer have access to $f(t - \theta)$ at time t , but this is implicitly represented in our compressed representation of the function: $f = \sum_{k=0}^{N-1} c_k(t) p_k(t)$. Thus we approximate $f(t - \theta)$ by $\sum_{k=0}^{N-1} c_k(t) p_k(t, t - \theta) = \sum_{k=0}^{N-1} c_k(t)$. Finally, this yields

$$\frac{d}{dt} c_n(t) = \frac{2\pi i n}{\theta} c_n(t) + \frac{1}{\theta} f(t) - \frac{1}{\theta} \sum_{k=0}^{N-1} c_k(t).$$

Hence $\frac{d}{dt} c(t) = Ac(t) + Bf(t)$ where

$$A_{nk} = \begin{cases} -1/\theta & \text{if } k \neq n \\ (2\pi i n - 1)/\theta & \text{if } k = n \end{cases}, \quad B_n = \frac{1}{\theta}.$$

Reconstruction At every time step t , we have

$$f(x) \approx \sum_n c_n(t) p_n(t, x) = \sum_n c_n(t) e^{2\pi i \frac{t-x}{\theta}}.$$

D.4.2 Fourier Recurrent Unit

Using the HiPPO framework, we can also derive the Fourier Recurrent Unit (FRU) [79].

Measure For each t , we will use a sliding measure uniform on $[t - \theta, t]$ and the basis $e^{2\pi i n \frac{x}{\theta}}$:

$$\omega(t, x) = \frac{1}{\theta} \mathbb{I}_{[t-\theta, t]}$$

$$p_n(t, x) = e^{2\pi i n \frac{x}{\theta}}.$$

In general the basis is not orthogonal with respect to the measure $\omega(t, x)$, but orthogonality holds at the end where $t = \theta$.

Derivatives

$$\frac{\partial}{\partial t} \omega(t, x) = \frac{1}{\theta} \delta_t - \frac{1}{\theta} \delta_{t-\theta}$$

$$\frac{\partial}{\partial t} p_n(t, x) = 0.$$

Coefficient Updates Plugging into equation 20 yields

$$\begin{aligned} \frac{d}{dt} c_n(t) &= \frac{1}{\theta} f(t) p_n(t, t) - \frac{1}{\theta} f(t - \theta) p_n(t, t - \theta) \\ &= \frac{1}{\theta} e^{2\pi i n \frac{t}{\theta}} f(t) - \frac{1}{\theta} e^{2\pi i n \frac{t}{\theta}} f(t - \theta). \end{aligned}$$

We no longer have access to $f(t - \theta)$ at time t , but we can approximate by ignoring this term (which can be justified by assuming that the function f is only defined on $[0, \theta]$ and thus $f(x)$ can be set to zero for $x < 0$). Finally, this yields

$$\frac{d}{dt} c_n(t) = \frac{e^{2\pi i n \frac{t}{\theta}}}{\theta} f(t).$$

Applying forward Euler discretization (with step size = 1), we obtain:

$$c_n(k+1) = c_n(k) + \frac{e^{2\pi i n \frac{k}{\theta}}}{\theta} f(t).$$

Taking the real parts yields the Fourier Recurrent Unit updates [79].

Note that the recurrence is independent in each n , so we don't have to pick $n = 0, 1, \dots, N-1$. We can thus pick random frequencies n as done in Zhang et al. [79].

D.5 Derivation for Translated Chebyshev

The final family of orthogonal polynomials we analyze under the HiPPO framework are the Chebyshev polynomials. The Chebyshev polynomials can be seen as the purely real analog of the Fourier basis; for example, a Chebyshev series is related to a Fourier cosine series through a change of basis [8].

Measure and Basis The basic Chebyshev measure is $\omega^{\text{cheb}} = (1-x^2)^{-1/2}$ on $(-1,1)$. Following Appendix B.1.3, we choose the following measure and orthonormal basis polynomials in terms of the Chebyshev polynomials of the first kind T_n .

$$\begin{aligned}\omega(t,x) &= \frac{2}{\theta\pi} \omega^{\text{cheb}} \left(\frac{2(x-t)}{\theta} + 1 \right) \mathbb{I}_{(t-\theta,t)} \\ &= \frac{1}{\theta\pi} \left(\frac{x-t}{\theta} + 1 \right)^{-1/2} \left(-\frac{x-t}{\theta} \right)^{-1/2} \mathbb{I}_{(t-\theta,t)} \\ p_n(t,x) &= \sqrt{2} T_n \left(\frac{2(x-t)}{\theta} + 1 \right) \quad \text{for } n \geq 1, \\ p_0(t,x) &= T_0 \left(\frac{2(x-t)}{\theta} + 1 \right).\end{aligned}$$

Note that at the endpoints, these evaluate to

$$\begin{aligned}p_n(t,t) &= \begin{cases} \sqrt{2} T_n(1) = \sqrt{2} & n \geq 1 \\ T_n(1) = 1 & n = 0 \end{cases} \\ p_n(t,t-\theta) &= \begin{cases} \sqrt{2} T_n(-1) = \sqrt{2}(-1)^n & n \geq 1 \\ T_n(-1) = 1 & n = 0 \end{cases}\end{aligned}$$

Tilted Measure Now we choose

$$\chi^{(t)} = 8^{-1/2} \theta \pi \omega^{(t)},$$

So

$$\frac{\omega}{\chi^2} = \frac{1}{\frac{\theta^2 \pi^2}{8} \omega} = \frac{8}{\theta\pi} \left(\frac{x-t}{\theta} + 1 \right)^{1/2} \left(-\frac{x-t}{\theta} \right)^{1/2} \mathbb{I}_{(t-\theta,t)}$$

which integrates to 1.

We also choose $\lambda_n = 1$ for the canonical orthonormal basis, so

$$g^{(t)} = p_n^{(t)} \chi^{(t)}$$

Derivatives The derivative of the density is

$$\frac{\partial}{\partial t} \frac{\omega}{\chi} = \frac{\partial}{\partial t} \frac{8^{1/2}}{\theta\pi} \mathbb{I}_{(t-\theta,t)} = \frac{8^{1/2}}{\theta\pi} (\delta_t - \delta_{t-\theta}).$$

We consider differentiating the polynomials separately for $n=0$, n even, and n odd, using equation (11).

Defined $z = \frac{2(x-t)}{\theta} + 1$ for convenience. First, for n even,

$$\begin{aligned}\frac{\partial}{\partial t} p_n(t,x) &= -\frac{2^{\frac{3}{2}}}{\theta} T_n' \left(\frac{2(x-t)}{\theta} + 1 \right) \\ &= -\frac{2^{\frac{3}{2}}}{\theta} T_n'(z) \\ &= -\frac{2^{\frac{3}{2}}}{\theta} \cdot 2n(T_{n-1}(z) + T_{n-3}(z) + \dots + T_1(z)) \\ &= -\frac{4n}{\theta} (p_{n-1}(t,x) + p_{n-3}(t,x) + \dots + p_1(t,x))\end{aligned}$$

For n odd,

$$\begin{aligned}
\frac{\partial}{\partial t} p_n(t,x) &= -\frac{2^{\frac{3}{2}}}{\theta} T_n' \left(\frac{2(x-t)}{\theta} + 1 \right) \\
&= -\frac{2^{\frac{3}{2}}}{\theta} T_n'(z) \\
&= -\frac{2^{\frac{3}{2}}}{\theta} \cdot 2n \left(T_{n-1}(z) + T_{n-3}(z) + \dots + T_1(z) + \frac{1}{2} T_0(z) \right) \\
&= -\frac{4n}{\theta} \left(p_{n-1}(t,x) + p_{n-3}(t,x) + \dots + 2^{-\frac{1}{2}} p_0(t,x) \right)
\end{aligned}$$

And

$$\frac{\partial}{\partial t} p_0(t,x) = 0.$$

Coefficient Dynamics

$$\begin{aligned}
c_n(t) &= \int f(x) p_n(t,x) \frac{2^{3/2}}{\theta\pi} \mathbb{I}_{(t-\theta,t)} dx \\
\frac{d}{dt} c_n(t) &= \int f(x) \frac{\partial}{\partial t} p_n(t,x) \frac{2^{3/2}}{\theta\pi} \mathbb{I}_{(t-\theta,t)} dx + \frac{2^{3/2}}{\theta\pi} f(t) p_n(t,t) - \frac{2^{3/2}}{\theta\pi} f(t-\theta) p_n(t,t-\theta) \\
&= -\frac{4n}{\theta} (c_{n-1} + c_{n-3} + \dots) + \frac{2^{3/2}}{\theta\pi} f(t) \begin{cases} \sqrt{2} & n \geq 1 \\ 1 & n = 0 \end{cases}
\end{aligned}$$

where we take $f(t-\theta) = 0$ as we no longer have access to it (this holds when $t < \theta$ as well).

In the usual way, we can write this as linear dynamics

$$\begin{aligned}
\frac{d}{dt} c(t) &= -\frac{1}{\theta} A c(t) + \frac{1}{\theta} B f(t) \\
A &= 4 \begin{bmatrix} 0 & & & \dots \\ 2^{-\frac{1}{2}} & 0 & & \\ 0 & 2 & 0 & \dots \\ 2^{-\frac{1}{2}} \cdot 3 & 0 & 3 & 0 \\ & \ddots & & \ddots \end{bmatrix} \\
B &= \frac{2^{3/2}}{\pi} \begin{bmatrix} 1 \\ \sqrt{2} \\ \sqrt{2} \\ \vdots \end{bmatrix}
\end{aligned}$$

Reconstruction In the interval $(t-\theta, t)$,

$$f(x) \approx \sum_{n=0}^{N-1} c_n(t) p_n(t,x) \chi(t,x).$$

E HiPPO-LegS Theoretical Properties

E.1 Timescale equivariance

Proof of Proposition 3. Let $\tilde{f}(t) = f(\alpha t)$. Let $c = \text{proj} f$ and $\tilde{c} = \text{proj} \tilde{f}$. By the HiPPO equation (18) update and the basis instantiation for LegS (equation (28)),

$$\begin{aligned}\tilde{c}_n(t) &= \langle \tilde{f}, g_n^{(t)} \rangle_{\mu^{(t)}} \\ &= \int \tilde{f}(t) (2n+1)^{\frac{1}{2}} P_n \left(2\frac{x}{t} - 1 \right) \frac{1}{t} \mathbb{I}_{[0,1]} \left(\frac{x}{t} \right) dx \\ &= \int f(\alpha t) (2n+1)^{\frac{1}{2}} P_n \left(2\frac{x}{t} - 1 \right) \frac{1}{t} \mathbb{I}_{[0,1]} \left(\frac{x}{t} \right) dx \\ &= \int f(\alpha t) (2n+1)^{\frac{1}{2}} P_n \left(2\frac{x}{\alpha t} - 1 \right) \frac{1}{\alpha t} \mathbb{I}_{[0,1]} \left(\frac{x}{\alpha t} \right) dx \\ &= c_n(\alpha t).\end{aligned}$$

The second-to-last equality uses the change of variables $x \mapsto \frac{x}{\alpha}$. \square

E.2 Speed

In this section we work out the fast update rules according to the forward Euler, backward Euler, bilinear, or generalized bilinear transform discretizations (cf. Appendix B.3). Recall that we must be able to perform matrix-vector multiplication by $I + \delta A$ and $(I - \delta A)^{-1}$ where δ is some multiple of the step size Δt (equation (13)).

It is easily seen that the LegS update rule involves a matrix A of the following form (Theorem 2): $A = D_1(L + D_0)D_2$, where L is the all 1 lower triangular matrix and D_0, D_1, D_2 are diagonal. Clearly, $I + \delta A$ is efficient (only requiring $O(N)$ operations), as it only involves matrix-vector multiplication by diagonals D_0, D_1, D_2 , or multiplication by L which is the cumsum operation.

Now we consider multiplication by the inverse $(I + \delta A)^{-1}$ (the minus sign can be absorbed into δ). Write

$$\begin{aligned}(I + \delta D_1(L + D_0)D_2)^{-1} &= (D_1(D_1^{-1}D_2^{-1} + \delta(L + D_0))D_2)^{-1} \\ &= \delta^{-1}D_2^{-1}(\delta^{-1}D_1^{-1}D_2^{-1} + D_0 + L)^{-1}D_1^{-1}\end{aligned}$$

Since diagonal multiplication is efficient, the crucial operation is inversion multiplication by a matrix of the form $L + D$.

Consider solving the equation $(L + D)x = y$. 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.

E.3 Gradient Norms

We analyze the discrete time case under the Euler discretization (Appendix B.3), where the HiPPO-LegS recurrent update is equation (4), restated here for convenience:

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

These gradient asymptotics hold under other discretizations.

We will show that

Proposition 7. *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), the dependence of $c_{\ell+1}$ on c_k and f_k, \dots, f_ℓ can be made explicit:

$$\begin{aligned} c_{\ell+1} &= \left(I - \frac{A}{\ell}\right) \dots \left(I - \frac{A}{k}\right) c_k \\ &\quad + \left(I - \frac{A}{\ell}\right) \dots \left(I - \frac{A}{k+1}\right) \frac{B}{k} f_k \\ &\quad + \left(I - \frac{A}{\ell}\right) \dots \left(I - \frac{A}{k+2}\right) \frac{B}{k+1} f_{k+1} \\ &\quad \vdots \\ &\quad + \left(I - \frac{A}{\ell}\right) \frac{B}{\ell-1} f_{\ell-1} \\ &\quad + \frac{B}{\ell} f_\ell. \end{aligned}$$

Therefore

$$\frac{\partial c_{\ell+1}}{\partial f_k} = \left(I - \frac{A}{\ell}\right) \dots \left(I - \frac{A}{k+1}\right) \frac{B}{k}.$$

Notice that A has distinct eigenvalues $1, 2, \dots, N$, since those are the elements of its diagonal and A is triangular (Theorem 2). Thus the matrices $I - \frac{A}{\ell}, \dots, I - \frac{A}{k+1}$ are diagonalizable with a common change of basis. The gradient then has the form $PDP^{-1}B$ for some invertible matrix P and some diagonal matrix D . Its norm is therefore bounded from below (up to constant) by the smallest singular value of P and $\|P^{-1}B\|$, both of which are nonzero constants, and the largest diagonal entry of D . It thus suffices to bound this largest diagonal entry of 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 $\log\left(1 - \frac{1}{x}\right)$. 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

$$\int \log\left(1 - \frac{1}{x}\right) = \int \log(x-1) - \log(x) = (x-1)\log(x-1) - x\log(x) = x\log\left(1 - \frac{1}{x}\right) - \log(x-1).$$

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

E.4 Function Approximation Error

Proof of Proposition 6. 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 $\langle \cdot, \cdot \rangle_{\mu^{(t)}}$ [14], by Parseval's identity,

$$\|f_{\leq t} - g^{(t)}\|_{\mu^{(t)}}^2 = \sum_{n=N}^{\infty} c_n^2(t).$$

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 B.1.1), integration by parts yields:

$$\begin{aligned}
c_n(t) &= \frac{(2n+1)^{\frac{1}{2}}}{2} \left[f\left(\frac{1+x}{2}t\right) \frac{1}{2n+1} (P_{n+1}(x) - P_{n-1}(x)) \right]_{-1}^1 \\
&\quad - \frac{(2n+1)^{\frac{1}{2}}}{2} \int_{-1}^1 \frac{t}{2} f' \left(\frac{1+x}{2}t \right) \frac{1}{2n+1} (P_{n+1}(x) - P_{n-1}(x)) dx.
\end{aligned}$$

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 supposed 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:

$$\begin{aligned} q_1 &= \frac{1}{2n+1} (P_{n+1} - P_{n-1}) = \frac{1}{n+O(1)} \cdot \frac{1}{2} (P_{n+1} - P_{n-1}) \\ q_2 &= \frac{1}{(n+O(1))^2} \frac{1}{2^2} (P_{n+2} - P_n - P_n + P_{n-2}) = \frac{1}{(n+O(1))^2} \frac{1}{2^2} (P_{n+2} - 2P_n + P_{n-2}) \\ q_3 &= \frac{1}{(n+O(1))^3} \frac{1}{2^3} (P_{n+3} - P_{n+1} - 2P_{n+1} + 2P_{n-1} + P_{n-1} - P_{n-3}) = \frac{1}{(n+O(1))^3} \frac{1}{2^3} (P_{n+3} - 3P_{n+1} + 3P_{n-1} - P_{n-3}) \\ &\dots \end{aligned}$$

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:

$$\|f_{\leq t} - g^{(t)}\|_{\mu^{(t)}}^2 = \sum_{n=N}^{\infty} c_n^2(t) = O(1)t^{2k} \sum_{n=N}^{\infty} \frac{1}{n^{2k}} = O(1)t^{2k} \frac{1}{N^{2k-1}}.$$

We then obtain that $\|f_{\leq t} - g^{(t)}\|_{\mu^{(t)}} = O(t^k N^{-k+1/2})$ as claimed. \square

Remark. 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 [45]. That result is also proved by integration by parts.

F Experiment Details and Additional Results

F.1 Model Architecture Details

Given inputs x_t or features thereof $f(x_t)$ in any model, the HiPPO framework can be used to memorize the history of features f_t through time. As the discretized HiPPO dynamics form a linear recurrent update similar in style to RNNs (e.g., Theorem 2), we focus on these models in our experiments. Thus, given any RNN update function $h_t = \tau(h_{t-1}, x_t)$, we simply replace the previous hidden state with a projected version of its entire history. Equations (31) lists the explicit update equations and Figure 6

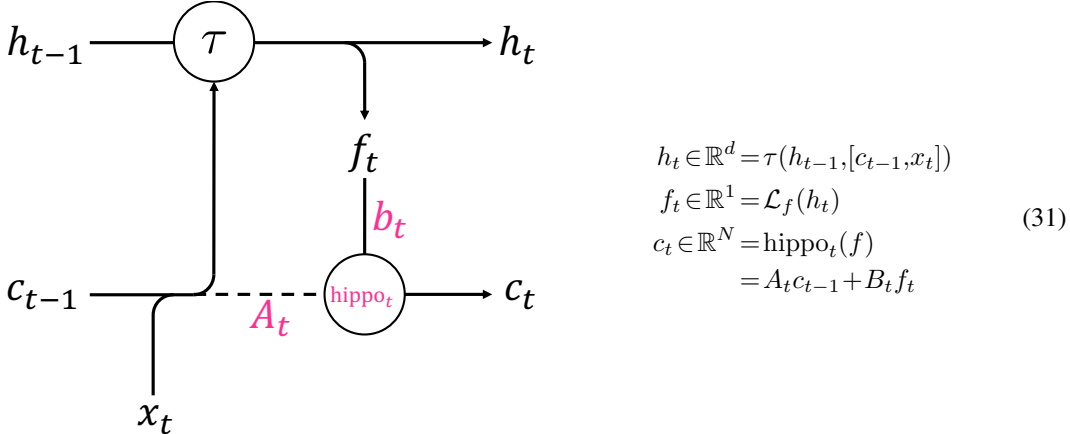


Figure 6: The simple RNN model we use HiPPO with, and associated update equations. \mathcal{L}_\square is a parametrized linear function, τ is any RNN update function, and $[\cdot]$ denotes concatenation. **hippo** is the HiPPO memory operator which orthogonalizes the history of the f_t features up to time t . A_t, B_t are fixed matrices depending on the chosen measure. N and d represent the approximation order and hidden state size, respectively.

illustrates the model. In our experiments, we choose a basic gated RNN update

$$\tau(h, x) = (1 - g(h, x)) \circ h + g(h, x) \circ \tanh(\mathcal{L}_\tau(h, x)), \quad g(h, x) = \sigma(\mathcal{L}_g(h, x)).$$

Methods and Baselines We consider the following instantiations of our framework HiPPO.

HiPPO-LegT, **LagT**, and **LegS**, use the translated Legendre, and tilted Laguerre, and scaled Legendre measure families with update dynamics (1), (2), and (3). As mentioned, LegT has an additional hyperparameter θ , which should be set to the timescale of the data if known a priori. We attempt to set it equal to its ideal value (the length of the sequences) in every task, and also consider θ values that are too large and small to illustrate the effect of this hyperparameter.

Our derivations in Appendices D.1 to D.5 show that there is a large variety of update equations that can arise from the HiPPO framework—for example, the tilted generalized Laguerre polynomials lead to an entire family governed by two free parameters (Appendix D.2)—many of which lead to linear dynamics of the form $\frac{d}{dt}c(t) = -Ac(t) + Bf(t)$ for various A, B . Given that many different update dynamics lead to such dynamical systems that give sensible results, we additionally consider the **HiPPO-Rand** baseline that uses random A and B matrices (normalized appropriately) in its dynamics.

We additionally compare against the following standard RNN baselines. The **RNN** is a vanilla RNN. The **MGU** is a minimal gated architecture, equivalent to a **GRU** without the reset gate. The HiPPO architecture we use is simply the MGU with an additional hippo intermediate layer. The **LSTM** is the most well-known and popular RNN architecture, which is a more sophisticated gated RNN. The **expRNN** [48] is the state-of-the-art representative of the *orthogonal RNN* family of models designed for long-term dependencies [3]. The **LMU** is the exact same model as in Voelker et al. [71]; it is equivalent to HiPPO-LegT with a different RNN architecture.

All methods have the same hidden size in our experiments. In particular, for simplicity and to reduce hyperparameters, HiPPO variants tie the memory size N to the hidden state dimension d . The hyperparameter N and d is also referred to as the number of hidden units.

Model and Architecture Comparisons The model (31) we use is a simple RNN that bears similarity to the classical LSTM and the original LMU cell. In comparison to the LSTM, HiPPO can be seen as a variant where the memory m_t plays the role of the LSTM’s hidden state and h_t plays the role of the LSTM’s gated cell state, with equal dimensionalities. HiPPO updates m_t using the fixed A transition matrix instead of a learned matrix, and also lacks “input” and “output” gates, so for a given hidden size, it requires about half the parameters.

The LMU is a version of the HiPPO-LegT cell with an additional hidden-to-hidden transition matrix and memory-to-memory transition vector instead of the gate g , leaving it with approximately the same number of trainable parameters.

Training Details Unless stated otherwise, all methods use the Adam optimizer [41] with learning rate frozen to 0.001, which has been a robust default for RNN based models [31, 71].

All experiments use PyTorch 1.5 and are run on a Nvidia P100 GPU.

F.2 Permuted MNIST

Task The input to the sequential MNIST (sMNIST) task [47] is an MNIST source image, flattened in row-major order into a single sequence of length 784. The goal of the model is to process the entire image sequentially before outputting a classification label, requiring learning long-term dependencies. A variant of this, the permuted MNIST (pMNIST) task, applies a fixed permutation to every image, breaking locality and further straining a model’s capacity for long-term dependencies.

Models are trained using the cross-entropy loss. We use the standard train-test split (60,000 examples for training and 10,000 for testing), and further split the training set with 10% to be used as validation set.

Baselines and Ablations Table 1 is duplicated here in Tables 4 and 5, with more complete baselines and hyperparameter ablations.

Table 4 consists of our implementations of various baselines related to our method, described in Appendix F.1. Each method was ran for 3 seeds, and the maximum average validation accuracy is reported. All methods used the same hidden size of 512; we found that this gave better performance than 256, and further increasing it did not improve more. All methods were trained for 50 epochs with a batch size of 100.

State of the Art Table 5 directly shows the reported test accuracy of various methods on this data (Middle and Bottom). Table 5 (Top) reports the test accuracy of various instantiations of our methods. We additionally include our reproduction of the LMU, which achieved better results than reported in Voelker et al. [71] (possibly due to a larger hidden size). We note that *all* of our HiPPO methods are competitive; each of them (HiPPO-LegT, HiPPO-LagT, HiPPO-LegS) achieves state-of-the-art among recurrent sequence models. Note that differences between our HiPPO-LegT and LMU numbers in Table 5 (Top) stem primarily from the architecture difference (Appendix F.1).

Timescale Hyperparameters Table 4 also shows ablations for the HiPPO-LegT and HiPPO-LagT timescale hyperparameters. HiPPO-LagT sweeps the discretization step size Δt (Section 2.4 and Appendix B.3). For LegT, we set $\Delta t = 1.0$ without loss of generality, as only the ratio of θ to Δt matters. These timescale hyperparameters are important for these methods. Previous works have shown that the equivalent of Δt in standard RNNs, i.e. the gates of LSTMs and GRUs (Section 2.4), can also drastically affect their performance [31, 66]. For example, the only difference between the URLSTM and LSTM in Table 5 is a reparametrization of the gates.

Table 4: **Our methods and related baselines.** Permuted MNIST (pMNIST) validation scores. (Top): Our methods. (Bottom): Recurrent baselines.

Method	Validation accuracy (%)
HiPPO-LegS	98.34
HiPPO-LagT $\Delta t = 1.0$	98.15
HiPPO-LegT $\theta = 200$	98.00
HiPPO-LegT $\theta = 2000$	97.90
HiPPO-LagT $\Delta t = 0.1$	96.44
HiPPO-LegT $\theta = 20$	91.75
HiPPO-LagT $\Delta t = 0.01$	90.71
HiPPO-Rand	69.93
LMU	97.08
ExpRNN	94.67
GRU	93.04
LSTM	92.54
MGU	89.37
RNN	52.98

Table 5: **Comparison to prior methods for pixel-by-pixel image classification.** Reported test accuracies from previous works on pixel-by-pixel image classification benchmarks. Top: Our methods. Middle: Recurrent baselines and variants. Bottom: Non-recurrent sequence models with global receptive field.

Model	Test accuracy (%)
HiPPO-LegS	98.3
HiPPO-Laguerre	98.24
HiPPO-LegT	98.03
LMU (ours)	97.29
URLSTM + Zoneout [46]	97.58
LMU [71]	97.15
URLSTM [31]	96.96
IndRNN [49]	96.0
Dilated RNN [10]	96.1
r-LSTM [69]	95.2
LSTM [31]	95.11
TrellisNet [6]	98.13
Temporal ConvNet [5]	97.2
Transformer [69]	97.9

F.3 Copying

Task In the Copying task [3], the input is a sequence of $L + 20$ digits where the first 10 tokens (a_0, a_1, \dots, a_9) are randomly chosen from $\{1, \dots, 8\}$, the middle N tokens are set to 0, and the last ten tokens are 9. The goal of the recurrent model is to output (a_0, \dots, a_9) in order on the last 10 time steps, whenever the cue token 9 is presented. Models are trained using the cross-entropy loss; the random guessing baseline has loss $\log(8) \approx 2.08$. We use length $L = 200$. The training and testing examples are generated in the same way.

Our motivation of studying the Copying task is that standard models such as the LSTM struggle to solve it. We note that the Copying task is much harder than other memory benchmarks such as the Adding task [3], and we do not consider those.

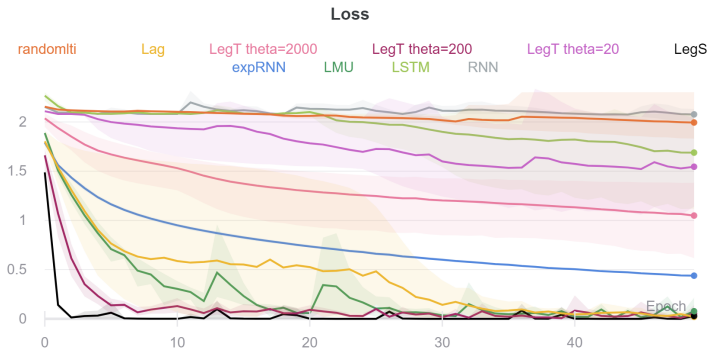


Figure 7: Loss on the Copying task. HiPPO methods are the only to fully solve the task. The hyperparameter-free LegS update is best, while methods with timescale parameters (e.g. LegT) do not solve the task if mis-specified.

Results The HiPPO-LegS method solves this task the fastest. The LegT method also solves this task quickly, only if the parameter θ is initialized to the correct value of 200. Mis-specifying this timescale hyperparameter to $\theta = 20$ or $\theta = 2000$ drastically slows down the convergence of HiPPO-LegT. The LMU (at optimal parameter $\theta = 200$) solves this task at comparable speed; like in Appendix F.2, differences between HiPPO-LegT ($\theta = 200$) and LMU here arise from the minor architecture difference in Appendix F.1.

The HiPPO-Rand baseline (denoted “random LTI” system here) does much worse than the updates with the dynamics derived from our framework, highlighting the importance of the precise dynamics (in contrast to just the architecture).

Standard methods such as the RNN and LSTM are also nearly stuck at baseline.

F.4 Trajectory Classification

Dataset The Character Trajectories dataset [4] from the UCI machine learning repository [25] 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 [34], GRU [17], and LMU [71]. Our implementations of these used 256 hidden units each.

The GRU-D [11] is a method for handling missing values in time series that computes a decay between observations. The ODE-RNN [61] and Neural CDE (NCDE) [40] 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. [40], 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.⁹

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 \rightarrow evaluation timescale shift of 100Hz \rightarrow 200Hz and 200Hz \rightarrow 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 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.

F.5 Online Function Approximation and Speed Benchmark

Task 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 2.1; the online approximation and reconstruction details are in Appendix C. The input function is randomly sampled from a continuous-time band-limited white noise process, with length 10^6 . The sampling step size is $\Delta t = 10^{-4}$, 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, 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 $L2$ loss. The HiPPO methods including LMU follow the fixed dynamics of Theorem 1 and Theorem 2.

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.

F.6 Sentiment Classification on the IMDB Movie Review Dataset

Dataset The IMDB movie review dataset [50] is a standard binary sentiment classification task containing 25000 train and test sequences, with sequence lengths ranging from hundreds to thousands of steps. The task is to classify the sentiment of each movie review into either positive or negative. We use 10% of the standard training set as validation set.

Methods RNN baselines include the LSTM [34], vanilla RNN, LMU [71], and expRNN [48]. Our implementations of these used 256 hidden units each.

Result As shown in Table 6, our HiPPO-RNNs have similar and consistent performance, on par or better than LSTM. Other long-range memory RNN approaches that constrains the expressivity of the network (e.g. expRNN) performs worse on this more generic task.

F.7 Mackey Glass prediction

The Mackey-Glass data [52] is a time series prediction task for modeling chaotic dynamical systems. We build on the implementation of Voelker et al. [71]. The data is a sequence of one-dimensional observations, and models are tasked with predicting 15 time steps into the future. The models are 4-layer stacked recurrent neural networks, trained with the mean squared error (MSE) loss. Voelker et al. [71] additionally consider a hybrid model with alternating LSTM and LMU layers, which improved on either by itself. We did not try this approach with our method HiPPO-LegS such as combining it with the LSTM or other HiPPO methods, but such ideas could further improve our performance. As a baseline method, the identity function does not simulate the dynamics, and simply guesses that the future time step is equal to the current input.

⁹Instead of the train timestamps being halved, equivalently the evaluation timestamps can be doubled.

Model	Test accuracy (%)
HiPPO-LegS	87.8 ± 0.2
HiPPO-LagT	88.0 ± 0.2
HiPPO-LegT $\theta=100$	87.4 ± 0.3
HiPPO-LegT $\theta=1000$	87.7 ± 0.2
HiPPO-LegT $\theta=10000$	87.9 ± 0.3
HiPPO-Rand	82.9 ± 0.3
LMU $\theta=1000$	87.7 ± 0.1
LSTM	87.3 ± 0.4
expRNN	84.3 ± 0.3
RNN	67.4 ± 7.7

Table 6: IMDB test accuracy, averaged over 3 seeds. Top: Our methods. Bottom: Recurrent baselines.

Fig. 8 plots the training and validation mean squared errors (MSE) of these methods. The table reports final normalized root mean squared errors (NRMSE) $\sqrt{\frac{\mathbb{E}[(Y-\hat{Y})^2]}{\mathbb{E}[Y^2]}}$ between the targets Y and predictions \hat{Y} . HiPPO-LegS outperforms the LSTM, LMU, and the best hybrid LSTM+LMU model from [68], reducing normalized MSE by over 30%.

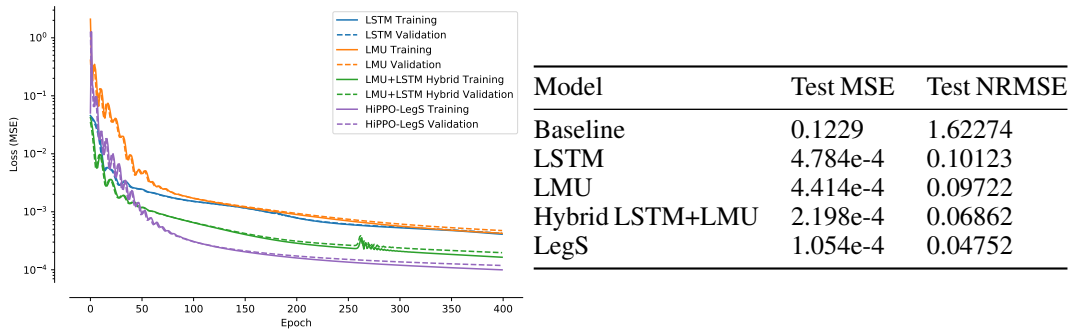


Figure 8: Mackey-Glass predictions

F.8 Additional Analysis and Ablations of HiPPO

To further analyze the tradeoffs of the memory updates derived from our framework, in Fig. 9 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 Appendix F.5. Given this function, we use the same methodology as in Appendix F.5 for processing the function online and then reconstructing it at the end.

In Figure 9(a, b), we plot the true function f , and its absolute approximation error based on LegT, LagT, 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 9(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 $x = 0$.

Generalized LagT Family Our LagT method actually comprises a family of related transforms, governed by two parameters α, β specifying the original measure and the tilting (Appendix D.2). Fig. 10 shows the error as these parameters change. Fig. 10(a) shows that small α generally performs better.

Fig. 10(b, c) show that the reconstruction is unstable for larger β , but small values of β work well. More detailed theoretical analysis explaining these tradeoffs would be an interesting question to analyze.

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 2.3 and 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.1), due to the nature of the sliding rather than scaling window.

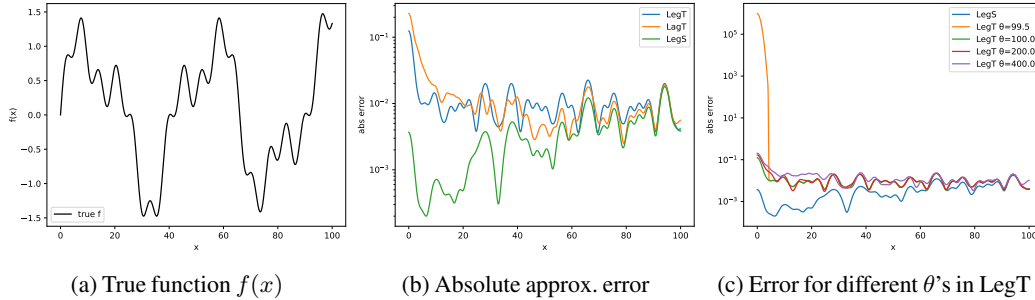


Figure 9: Function approximation comparison between LegT, LagT, and LegS. LegS has the lowest approximation error. LegT error is sensitive to the choice of window length θ , especially if θ is smaller than the length of the true function.

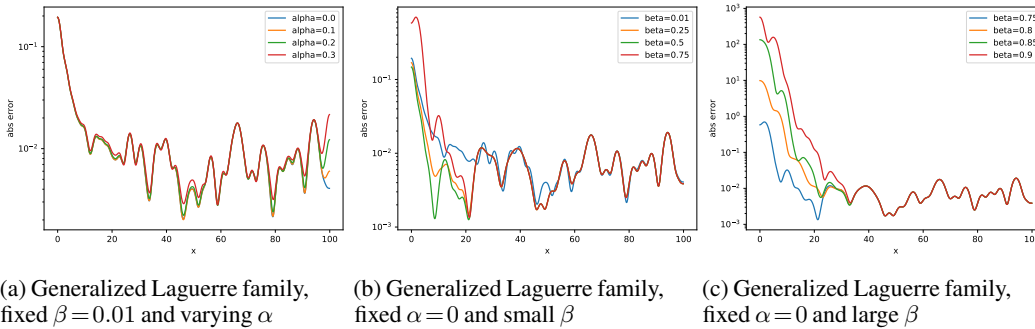


Figure 10: Function approximation comparison between different instantiations of the generalized tilted Laguerre family (Appendix D.2).