Wave Filtering for General Linear Dynamical Systems

Devan Shah
devan.shah@princeton.edu

Brandon Cho
brandon.cho@princeton.edu

December 12, 2024

Abstract. Within the past few years, state space models (SSMs) have become a valuable tool for sequence-modeling tasks, featuring prominently in both academic work and in industry applications [3; 4; 5; 7; 9; 11]. Notably, the Spectral Transform Unit (STU) proposed by Hazan et al. [9] relaxes state-space architecture through the introduction of spectral filters. The STU has achieved strong results and theoretical guarantees modeling symmetric linear dynamical systems and serves as a component in efficient post-transformer architectures for language. However, frequently linear dynamical systems in nature are not symmetric, and a large body of work (such as HiPPO [6]) has shown that systems capable of modeling non-symmetric linear dynamical systems are significantly more expressive. To that end, we propose alternate filter constructions to extend the STU to model general linear dynamical systems whose hidden-state transition matrices may not necessarily be symmetric.[1]

1 Introduction

1.1 Linear SSMs

Drawing inspiration from concepts in control theory, SSMs have positioned themselves as a contender to succeed transformers as the state-of-the-art approach for dealing with time-series data [5; 10; 11]. In ongoing research by the Hazan Lab, even the Linear SSM or slight relaxations of the Linear SSM, such as the STU, are incredibly expressive and work well within language models [1; 9; 11]. The Linear SSM is defined by the recurrence equations

$$\begin{aligned} x_t &= Ax_{t-1} + Bu_t \\ y_t &= Cx_t + Du_t \end{aligned} \tag{1}$$

which specify a discrete-time linear dynamical system (LDS) $f : u_t \mapsto y_t$, where x_t encodes the *hidden state* of the model and A, B, C, D specify the LDS [8]. Since the

[1]Code for our experimentation is available here: `https://github.com/dshah02/Filters_For_General_LDS` or in an interactive Google Colab environment here: `https://colab.research.google.com/drive/1mczJsXZBACE2KN_K-h4ItftPVU4ye-aS`. We are incredibly thankful to Windsor Nguyen and the Hazan Lab for sharing a Jupyter notebook STU instantiation and training script, which served as the template for our code. We provide a simplified STU implementation in our notebook and a similar full implementation is available at `https://github.com/windsornguyen/spectral_ssm`.

matrix D parametrizes a skip connection from u_t directly to y_t (bypassing the hidden state x_t) that can be learned through the other parameters A, B, C, much of the literature (including this paper) sets $D = 0$. As many systems in nature can be modeled as an LDS with suitably chosen parameters, much work has been done to find methods to extrapolate the matrices A, B, C given samples of time-sequence data obtained from natural systems [10]. We typically view x_t as the hidden state of the system, storing the information from prior inputs important for the future state of the system. When learning matrices A, B, C to fit a system, we need to fix a size of the hidden dimension and this often serves as a bottleneck on the representation capacity.

Note that, to better understand the dynamics of the Linear SSM, we can unroll the recurrence dynamics. With $*$ representing the convolution operator, note that:

$$\begin{aligned} y_t &= Cx_t + Du_t = C(Ax_{t-1} + Bu_t) + Du_t \\ &= Du_t + \sum_{i=1}^{t-1} CA^iBu_{t-i} \\ &= \langle D + CB, CAB, CA^2B, \dots \rangle * u \\ &= K * u \end{aligned} \tag{2}$$

Often times, when learning parameters to model a system with a Linear SSM, we achieves suitable performance with constrained structure on A. For instance, the S4 model architecture requires that A is a Normal Plus Low Rank (NPLR) matrix, which allows for easier parametrization and a faster learning process [7; 8; 12].

In particular, when A is known to be diagonal, which encompasses the case where A is symmetric since B, C can absorb orthonormal transformations, we have that:

$$A = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_d \end{bmatrix} \implies A^k = \begin{bmatrix} \lambda_1^k & 0 & \cdots & 0 \\ 0 & \lambda_2^k & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_d^k \end{bmatrix}$$

And thus, if we assume $D = 0$, and index the row vectors of B by b_j and the column vectors of C by c_j, Equation 2 can be extended as:

$$\begin{aligned} y_t &= \sum_{i=1}^{t-1} \bar{C}A^i\bar{B}u_{t-i} = \sum_{i=1}^{t-1}\sum_{j=1}^{k} c_j\lambda_j^i b_j u_{t-i} \\ &= \sum_{i=1}^{t-1}\sum_{j=1}^{k} c_j b_j \lambda_j^i u_{t-i} = \sum_{j=1}^{k} c_j b_j \sum_{i=1}^{t-1} \lambda_j^i u_{t-i} \\ &= \sum_{j=1}^{k} c_j b_j \sum_{i=1}^{t-1} \mu(\lambda_j)(i) u_{t-i}, \end{aligned} \tag{3}$$

With $\mu(\alpha)$ being the exponential function with base α, i.e. $\mu(\alpha)(i) = \alpha^i$. As i is integral, we also express $\mu(\alpha)$ as the vector $(1, \alpha, \alpha^2, \cdots)$ with i then referring to the index.

1.2 The Spectral Transform Unit

In the seminal paper [9], Hazan. et. al. find a sequence of vectors $\Phi = (\phi_1, \phi_2, \dots)$, which they call wave filters or spectral filters, so that that, with the first k filters and

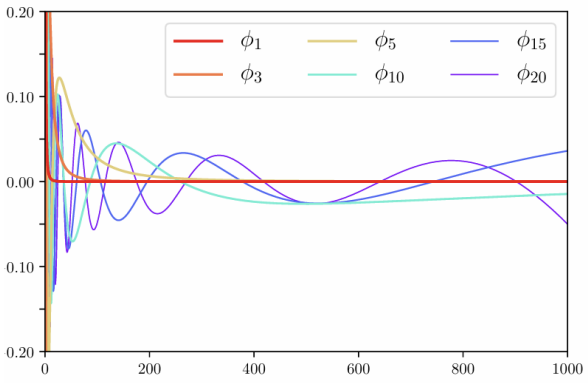

Figure 1: The entries of some of the wave filters. The x-axis is the time domain [9]

.

for all $\alpha \in [0,1]$, $\mu(\alpha)$ can be represented as a linear combination of $\phi_1, \ldots, \phi_k$ with error bounded by Ce^{-k}, with C independent of k or α. Thus, with the first 20 filters, $\phi_1, \ldots, \phi_{20}$, any vector $\mu(\alpha)$ can be represented with minimal error. This includes cases where $\alpha \approx 1$, which are long memory systems that are typically more challenging to learn. More specifically, by Lemma 4.1 in [9], where T denotes the length of the input sequences:

Lemma 1 *Choose any $\alpha \in [0,1]$. Let $\tilde{\mu}(\alpha)$ be the projection of $\mu(\alpha)$ onto the k-dimensional subspace of $\mathbb{R}^T$ spanned by $\{\phi_j\}_{j=1}^k$. Then, with absolute constant $c_0 > 3.4$,*

$$\|\mu(\alpha) - \tilde{\mu}(\alpha)\|^2 \leq O\left(c_0^{-k/\log T}\sqrt{\log T}\right).$$

The wave filters $\{\phi_j\}_{j=1}^k$ are chosen as the top k eigenvectors of the Hankel matrix $Z = \int_0^1 \mu(\alpha) \otimes \mu(\alpha) \in \mathbb{R}^{T \times T}$ with entries

$$Z_{ij} = \frac{2}{(i+j)^3 - (i+j)}.$$

Thus, with $M^{(j)} = \sum_{l=1}^d \langle \phi_j, \mu(\alpha_l) \rangle (c_j b_j)$, block matrix $M_\Theta = [M^{(1)}, \ldots, M^{(k)}]$, and matrix X with $X_{ij} = \sum_{q=1}^{T-1} \phi_j(q) u_{t-q}(i)$, representing convolutions between the input dimensions and the filters:

$$y \approx M_\Theta X \tag{4}$$

We can visualize the filters in Figure 1.

The larger intuition is that, by the final line of Equation 3, since $\mu(\lambda_j)$ is well represented by Φ, we can represent the convolution $\sum_{i=1}^{t-1} \mu(\lambda_j)(i) u_{t-i}$ as a weighted sum of convolutions between the wave filters and the inputs. These can be pre-computed, and multiplied by a suitable matrix to approximate the system. More loosely, representing $\phi_i * u$ as a coordinate wise convolution, we can equivalently express Equation 4 by $y = P \sum_{i=1}^k \phi_i * u$ for some P. Thus, to learn a linear dynamical system, rather than learning the matrices A, B, C, we can learn a strong approximation by simply learning the matrix M. In practice, with initial data drawn from a symmetric LDS, gradient descent on the matrix M will fit the system quickly and accurately, with loss typically dropping below 10^{-4} for stable systems.

In the initial implementation [9], Hazan et. al. additionally add scaling terms to each filter, but this is largely a design choice for stability. For vector ϕ_i, the scaling term $\sigma_i^{1/4}$

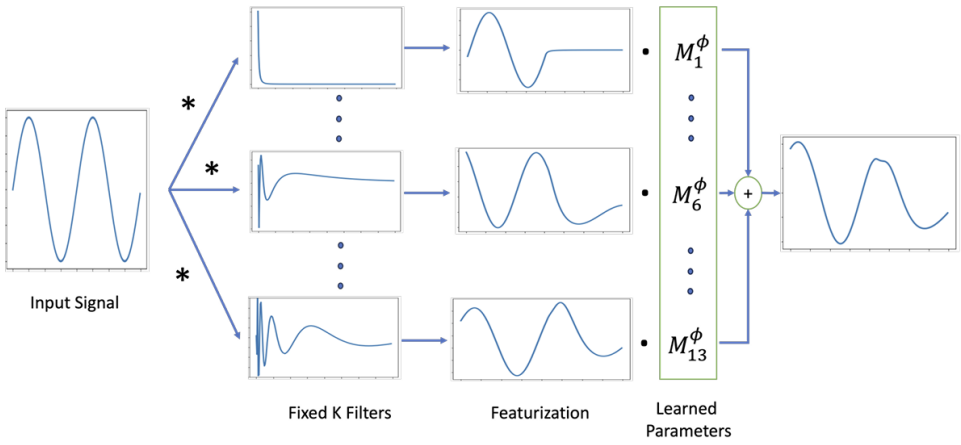

Figure 2: Viewpoint of the spectral parametrization as learning coefficients on the fixed filters [9]

is used where σ_i is the corresponding eigenvalue of Z for the eigenvector ϕ_i. In some implementations of the STU, there additionally is an auto-regressive matrix component and additional skip-connections, but later implementations have had these as optional parameters and we omit them for simplicity [1; 9]. As is commonly done, we omit the norm limitation of M included in [9].

Thus, we present the following gradient descent-based algorithm to learn the matrix M for the STU [9]. It is not important to work through each detail – the key observation is that as the symmetric LDS convolutional kernel approximately lies in low-dimensional space, and as we have found vectors that serves as "basis" for this space, we have a different technique for learning symmetric linear dynamical systems.

Algorithm 1 Wave-filtering algorithm for LDS sequence prediction

Require: filter parameter k, learning rate η, dataset $\mathcal{D} = (u^{(i)}, y^{(i)})_{i=1}^{m}$ with $u^{(i)}$ and $y^{(i)}$ being input and output sequences, input dimension n, output dimension r

Compute $\{(\sigma_j, \phi_j)\}_{j=1}^{k}$, the top k eigenpairs of Z_T.
Initialize $M_1 \in \mathbb{R}^{r \times nk}$,
for $t = 1, \ldots, m$ **do**
    Initialize $\mathcal{L} = 0$
    for $\ell = 1, \ldots, L$ **do** ###predict output at time ℓ given prior inputs
        Compute $\mathbf{X_t} \in \mathbb{R}^{nk}$, with entries $X_{(i,j)} := \sigma_j^{1/4} \sum_{q=1}^{\ell-1} \phi_j(q) u_{\ell-q}^{(t)}(i)$.
        Predict $\hat{\mathbf{y}}_\ell = M_t \mathbf{X_t}$
        $\mathcal{L} = \mathcal{L} + \|\mathbf{y}_\ell^{(\mathbf{t})} - \hat{\mathbf{y}}_\ell\|^2$
    end for
    Gradient update: $M_{t+1} \leftarrow M_t - \eta \nabla_{M_t} \mathcal{L}$
end for

We note that this is different from the online algorithm algorithm presented in [9] but in line with more recent implementations [1; 11].

1.3 Limitations of the STU

Although the Spectral Transform Unit performs well at learning symmetric linear dynamical systems, by its design, it is limited when learning non-symmetric linear dynamical systems. Unfortunately, many tasks are much easier to learn given a model with the capacity to learn a broader class of linear dynamical systems.

More specifically, the base STU model is effective at modeling linear dynamical systems with diagonal A, since the Hankel matrix-derived $\{\phi_j\}_{j=1}^{k}$ are able to approximate $\mu(\alpha)$ with error exponential in k. However, when A is not diagonal we no longer have the same theoretical guarantees as in Lemma 1, and there may be alternative sets of filters that outperform the Hankel matrix eigenvectors in this more general setting. As far as intuition on why that may be the case and what concerns arise, for non-symmetric matrix A, first note that A^i for large i will likely be dominated by a few large eigenvalues and thus act similar as a symmetric matrix. Thus, we expect the STU to correctly model the distant input contributions for this LDS. However, we expect the recent contributions to follow more complex patterns that will not be modelable by Φ. As an example, Figure 3 showcases the kernel K of the following randomly generated LDS with eigenvalues 0.99 and -0.67:

$$A = \begin{bmatrix} 0.1140 & 1.1414 \\ 0.6024 & 0.2052 \end{bmatrix}, \quad B = \begin{bmatrix} 0.3968 \\ 0.1338 \end{bmatrix}, \quad C = \begin{bmatrix} 0.3993 & 0.2379 \end{bmatrix}$$

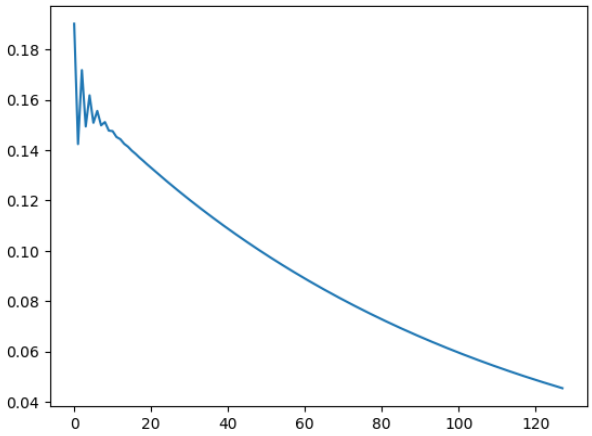

Figure 3: Kernel K of the specified LDS. Note how, long-term dynamics are dominated by the larger eigenvalue and are exponential, yet short term dynamics may be more complicated.

To address a similar issue, Gu et al. in [6] introduced high-order polynomial projection operators (*HiPPO matrices*) that allow the hidden state x_t to parametrize the inputs u_t in the set of Legendre polynomials, which form an orthonormal basis for the Hilbert space $L^2([-1, 1])$ and can uniformly approximate any continuous function on $[-1, 1]$ by the Weierstrass approximation theorem. These HiPPO matrices allow for the SSM to begin to overcome the memory horizon issues faced by traditional RNNs.

We anticipate that with certain learned filters, we can extend the Spectral Transform Unit to model a more complicated set of dynamics, such as linear dynamical systems governed by a non-symmetric matrix A. We hope that, based on the success of the HiPPO matrices, which are symmetric plus low rank, even a slightly more expressive set of filters can lead to large improvements.

2 Methodology

2.1 Overview

The Spectral Transform Unit can effectively represent symmetric linear dynamical systems, since under a linear transformation, the kernel of the linear dynamical system can be well represented by the spectral filters Φ. Thus, to extend the STU to better model non-symmetric linear dynamical systems, we will attempt to find filters that can well represent the kernels of non-symmetric linear dynamical systems. To do so, we will generate many non-symmetric linear dynamical systems and attempt to find filters useful in fitting all of them simultaneously, with the goal that our filters generalize to the distribution we are sampling from. We test out many methods of parameterizing the filters and test their performance against the spectral filters.

2.2 Experimentation

In order to extend the STU, we take the wave filters $\{\phi_j\}_{j=1}^k$ specified by Hazan et al., which we call *fixed filters* or *spectral filters*, and append to them a set of *adaptive filters* $\{\psi_i\}_{i=1}^m$ that are learned through gradient descent. At times, we also refer to the *adaptive filters* as *learned filters*. Our training methodology is designed so that the adaptive filters, serving as a basis, minimize the expected representation error of the kernels of linear dynamical systems drawn from a certain distribution, which is described below. Let d_h denote the dimension of the hidden state, d_u denote the dimension of the input, d_o denote the dimension of the output, and T denote the input sequence length.

In order to learn the filters $\{\psi_i\}_{i=1}^m$, we select some small $\delta > 0$ and randomly generate $L = 50,000$ linear dynamical systems $\{(A_\alpha, B_\alpha, C_\alpha)\}_{\alpha=1}^L$ by first sampling the entries of a $d_h \times d_h$ random matrix $A'_{\alpha,ij} \sim \mathcal{N}(0,1)$ and the coefficients $X_\alpha \sim \text{Unif}(0,1)$ independently for each α, i, j and then defining

$$\begin{aligned} A_{\alpha,ij} &= X_\alpha \frac{1-\delta}{\lambda_{max}(A'_\alpha)} A'_{\alpha,ij} \\ B_{\alpha,ij} &\sim \text{Unif}(0,1) \\ C_{\alpha,ij} &\sim \text{Unif}(0,1) \end{aligned} \tag{5}$$

for appropriate ranges of i, j specified by the dimensions of $A_\alpha \in \mathbb{R}^{d_h \times d_h}$, $B_\alpha \in \mathbb{R}^{d_h \times d_u}$, and $C_\alpha \in \mathbb{R}^{d_o \times d_h}$. This process directly bounds the maximum eigenvalue of A_α by $X_\alpha(1-\delta)$. Other distributions that sample the entire desirable space of systems should also suffice.

We will have a learned parameter M_α for each task $\alpha \in \{1, \ldots, 50000\}$ and all of the tasks will share the same set of adaptive filters, which will also be updated by gradient descent. To minimize error when learning the linear dynamical systems, the adaptive filters will need to parameterize a basis that minimizes the kernel representation error across the $50,000$ linear dynamical systems. As the adaptive filters are represented with few parameters (i.e. as sinusoids and by the mechanisms below) and we train with relatively few epochs, we expect that they will not overfit to the $50,000$ training systems, but will instead minimize the expected kernel representation error for an LDS chosen by our sampling procedure. As these LDSs have complex behavior and have non-symmetric A, we thus expect the adaptive filters to model this behavior.

To better understand how initialization and parametrization of our adaptive filters affects the performance of our modified STU, we parameterize and initialize $\{\psi_i^{(0)}\}_{i=1}^m$ using one of the following procedures:

1. *Sinusoidal initialization*, where each $\psi_i^{(0)}(t) = \beta_i \sin\left(\frac{2\pi\omega_i t}{T} + \frac{X_i}{T}\right) \exp\left(-\frac{2\gamma_i t}{T}\right)$ for $X_i, \beta_i, \gamma_i, \omega_i \sim \mathcal{N}(0,1)$ i.i.d. is the product of a randomly modified sine curve and an exponential decay term,

2. *Triple sinusoidal initialization*, where each $\psi_i^{(0)}(t)$ is the product of three sine curves initialized as in (1) with an exponential decay term $\exp\left(-\frac{2\gamma_i t}{T}\right)$, and

3. *Neural network parametrization*, where each $\psi_i^{(0)}(t)$ is a neural network with input dimension $N_0 = 1$, depth D with varying hidden layer sizes, and output dimension $N_D = 1$. We additionally added a learned decay parameter as with the other initialization methods.

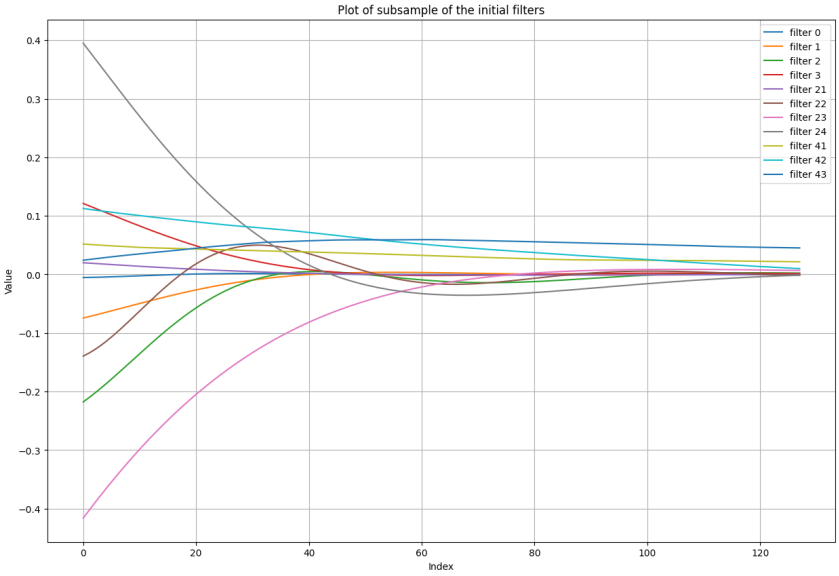

Figure 4: Sampled initializations for all three methods. Filters 0 through 3 are sinusoidal, filters 21 through 23 are triple sinusoidal, and filters 41 through 43 are neural network parametrized.

These initialization options were chosen for their wave-like shapes and their relative expressivity for continuous functions on $[0, T]$. Notably, each parametrization has significantly fewer than 50,000 parameters per filter[2] and thus we do not expect the filters to overfit to the training set.

We also attempted to use *Gaussian initialization*, where $\mu_i, v_i, C_i \sim \mathcal{N}(0,1)$ i.i.d. parameterize a Gaussian distribution $\psi_i^{(0)}(t) = C_i \exp\left(-(\frac{t}{T} - \mu_i)^2/(2|v_i|)\right)$, and *polynomial initialization*, where each $\psi_i^{(0)}(t) = \sum_{j=0}^{7} X_{i,j}(\frac{t}{T})^j$ is a degree seven polynomial with i.i.d. random coefficients $X_{i,j} \sim \mathcal{N}(0,1)$, but found that neither method was able to sufficiently represent the randomly generated LDSs. We would also like to remark that, although it may seem that the bases are excessive (i.e., only 8 terms are sufficient to represent degree 7 polynomials, yet we have more than 8 filters), even linearly dependent basis vectors can

[2]All methods tested except the neural network have fewer than 10 parameters per filter.

be useful by more densely sampling the filter space, as gradient descent is an imperfect optimizer.

After initialization, we run the following minibatch gradient descent algorithm to optimize both $\{\psi_i^{(\tau)}\}_{i=1}^m$ (where τ denotes iterations) and the matrices $M_\alpha^{(\tau)}$ associated with each task:

Algorithm 2 Gradient descent for $\{\psi_i^{(\tau)}\}_{i=1}^m$ and M_α

Require: learning rate η, number of iterations ν, batch size B

1: **for** $\tau = 1, \ldots, \nu$ **do**
2: Sample a batch $\{(A_\beta, B_\beta, C_\beta)\}_{\beta=1}^B$ of size B from the tasks $\{(A_\alpha, B_\alpha, C_\alpha)\}_{\alpha=1}^L$
3: Sample a random input $u_\beta \in \mathbb{R}^{B \times T \times d_u}$ with entries i.i.d. standard Gaussian $\mathcal{N}(0, 1)$
4: Generate trajectories $y_\beta(t)$ for each LDS with initial state u_β
5: Generate predictions $\hat{y}_\beta(t)$ using the learned $M_\beta^{(\tau)}$ and filters $\{\phi_j\} \cup \{\psi_i^{(\tau)}\}$ (alternatively, just use the adaptive filters $\{\psi_i^{(\tau)}\}$)
6: Compute MSE loss $\mathcal{L} = \frac{1}{2B} \sum_{\beta=1}^B \|y_\beta - \hat{y}_\beta\|^2$
7: Gradient update $M_\beta^{(\tau+1)} \leftarrow M_\beta^{(\tau)} - \eta \nabla_{M_\beta} \mathcal{L}$ and $\psi_i^{(\tau+1)} \leftarrow \psi_i^{(\tau)} - \eta \nabla_{\psi_i} \mathcal{L}$
8: **end for**

We also attempted to use two separate learning rates for the matrices M_α and the filters $\{\psi_i\}$, but ran into stability issues with gradient descent. We performed our training primarily on an NVIDIA A100 GPU with 40GB of GPU memory allocated, with some training also conducted via Princeton University's Della cluster.

We hypothesize that our model performs better on shorter sequences than the STU, since we expect short term dynamics to not follow exponential patterns for the reasons described in the limitations section.

3 Results and Discussion

3.1 Combined Fixed and Adaptive Filters

We first evaluate the win rate and mean error of the combined set of fixed and adaptive filters against just the fixed filters found in the base STU model in the setting $d_u = 1$, $d_h = 2$, $d_o = 1$ and the same LDS generation procedure as above. Using 60 adaptive filters and 60 fixed filters (mean error 0.101) achieves a 48% win rate against using 120 fixed filters as in the base STU (mean error 0.092). Using 30 adaptive filters and 30 fixed filters (mean error 0.107) achieves a 65% win rate against using 60 fixed filters as in the base STU (mean error 0.112).

3.2 Comparison with STU Performance

We also evaluate our methodology with that of the base STU by comparing the MSE of the STU with fixed filters $\{\phi_j\}_{j=1}^m$ with the MSE of the STU leveraging our adaptive filters $\{\psi_j\}_{j=1}^m$ and no fixed filters. To make this comparison, we chose $N = 20$ randomly-sampled linear dynamical systems and trained the models with $\nu = 1,000$ epochs of training on Gaussian inputs with length T (with no batches). To evaluate performance, we sample additional random input sequences of length T after training, and compare the MSE of

the base STU (with fixed filters) and our adaptive filters with respect to the outputs $y_1, \ldots, y_T$ generated by the original N LDSs to derive win rate statistics.

First, for input lengths $T = 16, 32$ we evaluate with $m = 15$ filters, and for $T = 64, 128, 256, 512$ we evaluate with $m = 60$ filters (split equally among each of the three initialization methods), both with diagonal A and dimensions $d_u = 1$, $d_h = 2$, $d_o = 1$. This yields the win rate table in Figure 5 below.

T	adaptive filter win rate
16	0.45 (9/20)
32	0.45 (9/20)
64	0.30 (6/20)
128	0.30 (6/20)
256	0.30 (6/20)
512	0.20 (4/20)

Figure 5: Win rates for adaptive filter method over base STU with diagonal LDSs.

These results are expected, since the STU filters were constructed to achieve loss decreasing exponentially with the number of filters, while the primary aim of our methodology was to generate filters that perform better on non-symmetric linear dynamical systems. It is interesting to note that that the win-rate decreases with the sequence length. We expect that this is due to challenges when training filters as the filters need to learn to set later terms to 0 to minimize loss (as far away terms have no impact in long sequences), but the functions that parameterize our filters may struggle to go to zero or the pressure towards zero may destabilize the filters.

We then repeated these experiments for general (non-symmetric) linear dynamical systems, with $T = 128, 256$, $m = 60$ filters (also split evenly among each initialization method), and $N = 20$. This yields the win rates in Figure 6.

These results indicate that our learned filters do not outperform the spectral filters directly on non-symmetric linear dynamical systems; further discussion on this can be found in the next section. A subsample of the m adaptive filters for the $T = 256$ model can be seen in Figure 7, and an example of our adaptive filters fitting one of the non-symmetric linear dynamical systems can be seen in Figure 8.

T	adaptive filter win rate
128	0.25 (5/20)
256	0.30 (6/20)

Figure 6: Win rates for adaptive filter method over base STU with general LDSs. Unfortunately, due to a system error on the machine, we were unable to test the other lengths.

3.3 Discussion

Our results are largely inconclusive. Although our learned filters alone underperform compared to the spectral filters on symmetric and non-symmetric systems, using both spectral filters and learned filters outperforms using more spectral filters in some cases.

With proper parametrization and initialization of the learned filters, as the neural network parametrization can learn an approximation of the spectral filters, this work

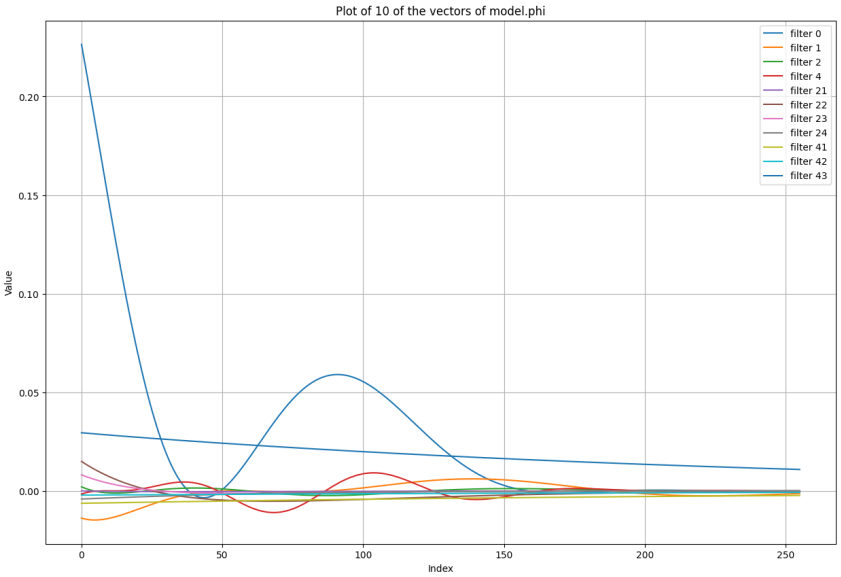

Figure 7: Sample learned filters across all three initialization methods (see Figure 3 for labels). These filters were normalized for plotting.

should be a strict generalization of the STU [9]. Thus, the fact that head-to-head competitions between the learned and spectral filters tilts disproportionately towards the spectral filters suggests an issue with our parametrization or learning approach. Generally, the results above indicate that there is significant room for improvement in our adaptive filter approach. We had tried fully parameterized filters and had poor performance, but it is possible that with more epochs and a more intensive training procedure, these filters could perform well.

We hypothesize that initialization is incredibly important to the performance of the filters, and our testing supports this idea. Thus another direction for future work includes initializing the adaptive filters to be near the the spectral filters and learning from there. Additionally, we hope to explore mathematical results on the class of kernels that arise with non-symmetric linear dynamical systems, hoping that perhaps a mathematical construction for filters exist similar to the symmetric case.

One potential limitation of our work is that our method for generating linear dynamical systems (A, B, C) in Equation 5 is not necessarily representative of many real-world systems, in which A contains many eigenpairs with large eigenvalues (> 0.9). For example, because our matrix A'_α is drawn from the Ginibre ensemble, it is well known that as $d_h \to \infty$ the distribution of the eigenvalues of $\frac{1}{\sqrt{d_h}} A'_\alpha$ converges to the uniform distribution on the complex unit disc, so only a small proportion of the eigenvalues of A_α will have norm greater than 0.9 [2].

4 Conclusion

We observe that learned adaptive filters in conjunction with fixed filters can be competitive with the original spectral filters described in Hazan et al [9]. However, we have not yet found filters that improve upon the spectral filters for non-symmetric linear dynamical systems, suggesting that – for systems from our generating process – they are still well modeled by a symmetric linear SSM. However, we expect that real-world systems abide by

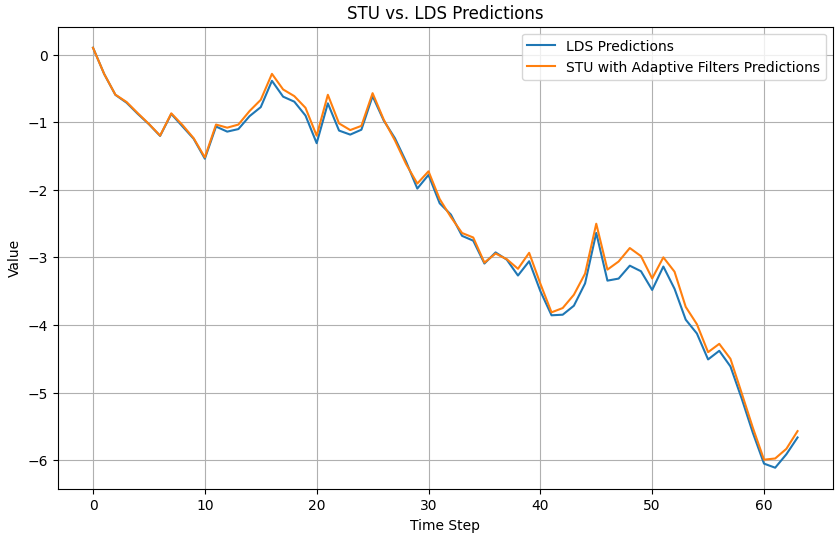

Figure 8: An example of the adaptive filters fitting a non-symmetric LDS.

different generative patterns and distributions, and we hope in the future to investigate whether our observed patterns hold in real-world data.

Additionally, its worth emphasizing that the methods employed in this paper are data-independent. Thus, these techniques can be easily scaled and, for future work, we expect to try similar methods at a larger scale. We are optimistic that learned filters represent a path forward from the spectral filters, and we look forward to future work investigating this area.

References

[1] Agarwal, N., Suo, D., Chen, X., and Hazan, E. Spectral state space models, 2024.

[2] Bai, Z. D. Circular law. *Ann. Probab. 25*, 1 (1997), 494–529.

[3] Bourdois, L. Introduction to state space models (ssm). `https://huggingface.co/blog/lbourdois/get-on-the-ssm-train`, July 2024.

[4] Cartesia. Real-time multimodal intelligence for every device. `https://www.cartesia.ai`. Accessed: 2024-12-12.

[5] Gu, A., and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces, 2024.

[6] Gu, A., Dao, T., Ermon, S., Rudra, A., and Re, C. Hippo: Recurrent memory with optimal polynomial projections, 2020.

[7] Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces, 2022.

[8] Gu, A., Goel, K., Saab, K., and Ré, C. Structured state spaces: Combining continuous-time, recurrent, and convolutional models. *Hazy Research Blog* (Jan 2022).

[9] Hazan, E., Singh, K., and Zhang, C. Learning linear dynamical systems via spectral filtering, 2017.

[10] Kalman, R. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering 82*, Series D (1960), 35–45.

[11] Liu, Y. I., Nguyen, W., Devre, Y., Dogariu, E., Majumdar, A., and Hazan, E. Flash stu: Fast spectral transform units, 2024.

[12] Rush, S., and Karamcheti, S. The annotated s4: Efficiently modeling long sequences with structured state spaces. `https://srush.github.io/annotated-s4/`, 2024.