A Survey of State Space Models: From Linear Systems to Language

Devan Shah and Brandon Cho
Princeton University, Class of 2026

1 Overview

In this paper, we will survey the development and literature behind State Space Models (SSMs), a model architecture and approach inspired by control theory that is now poised to succeed transformers for sequence modeling tasks. We will first discuss the origins of SSMs as linear dynamical systems, how SSMs extend existing deep sequence-to-sequence architectures, and the development of deep architectures around SSMs. We will also discuss recent research and results from Princeton faculty in State Space Models, such as Mamba [7] and Spectral Transformers [5].

The exposition of our paper will largely follow the exposition in the dissertations of Albert Gu [6] and Tri Dao [3], with some topic inspiration from the Huggingface Community Blog on the topic [2] and the foundational S4 paper [10] and covering interesting results along the way.

2 Linear Dynamical Systems and Background

2.1 The Continuous SSM

A State Space Model is designed for the task of sequence-to-sequence prediction, which involves modeling an unknown function $f : u(t) \in \mathbb{R}^n \to y(t) \in \mathbb{R}^m$, where $u(t)$ is the input function and $y(t)$ the output function. For any practical use case, $u(t)$ and $y(t)$ are discretized to sequences. The state space model aims to approximate f with the differential equations:

$$\begin{aligned} x'(t) &= A_t x(t) + B_t u(t) \\ y(t) &= C_t x(t) + D_t u(t) \end{aligned} \tag{1}$$

Where A_t, B_t, C_t, D_t are matrices of the appropriate dimension and characterize the SSM, and $x(t) \in \mathbb{R}^h$ is known as the hidden state. We can view $x(t)$ as storing the state of the system and all necessary context for determining y_t. The matrices A_t, B_t, C_t, D_t may have time-dependent value, but it is more common to consider the time-invariant SSM, which has fixed A, B, C, D. As the update equations in Eq. 1 are linear, we refer to a system modeled by Eq. 1 as a linear dynamical system (LDS).

SSMs are broadly used in within control theory and are related to Hidden Markov Models [10, 15]. We also note that many authors use the term State Space Model differently. In some exposition the linear dynamical system is referred to as the linear SSM, and State Space Models are used to refer to any sequence-to-sequence model formulated recurrently with a hidden state. In this exposition, we will use SSM as referring to a learned model based on the update equations Eq. 1, and an LDS as a general system obeying Eq. 1, the SSM update equation.

2.2 The Discrete SSM

In practice we typically sample signals at fixed intervals Δ, and can then approximate the (time-invariant) continuous SSM with the discrete or recurrent SSM, which has recurrence equations [11]:

$$\begin{aligned} x_t &= \bar{A} x_{t-1} + \bar{B} u_t \\ y_t &= \bar{C} x_t + \bar{D} u_t \end{aligned} \tag{2}$$

Where, $\bar{A}, \bar{B}, \bar{C}, \bar{D}$ are defined by:

$$
\begin{aligned}
\bar{A} &= e^{\Delta A} \\
\bar{B} &= A^{-1} e^{(\Delta A - I)} B \\
\bar{C} &= C \\
\bar{D} &= D
\end{aligned} \tag{3}
$$

Note that as $\Delta \to 0$, we recover the initial continuous SSM.

Despite the simple evolutionary rules, state space models prove very expressive for natural systems and can be expanded upon to handle more complex systems, such as language [1].

2.3 Convolutional Representation

An additional benefit of the simple structure is that SSMs have a convolutional representation. Note that, with $*$ being the convolution operator:

$$
\begin{aligned}
y_t &= \bar{C}x_t + \bar{D}u_t = \bar{C}(\bar{A}x_{t-1} + \bar{B}u_t) + \bar{D}u_t \\
&= \bar{D}u_t + \sum_{i=1}^{t-1} \bar{C}\bar{A}^i \bar{B} u_{t-i} \\
&= \langle \bar{D} + \bar{C}\bar{B}, \bar{C}\bar{A}\bar{B}, \bar{C}\bar{A}^2\bar{B}, \dots \rangle * u \\
&= \bar{K} * u
\end{aligned} \tag{4}
$$

In this sense, an SSM can almost be viewed as a set of convolutional filters on an input sequence, lending itself to parallelization [19]. However, unlike traditional convolutional filters, such as in the CNN model [16], this convolution has infinite width. We can view the difference between the continuous, discrete, and convolutional representation in Fig. 1.

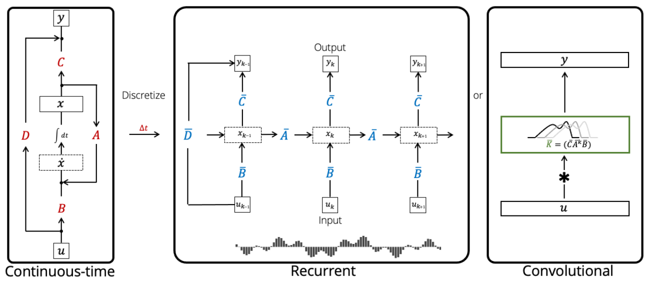


Figure 1: The multiple views of an SSM [11]

2.4 Considerations as a model

Despite the underlying operations being linear, the SSM empirically has strong representation capacity, as we will show in later results.

A very desirable trait in sequence-to-sequence modeling is the ability to model long range dependencies. Typically, for the SSM, we assume $||\bar{A}||_2 \leq 1$ where $||\cdot||_2$ is the spectral norm, as otherwise the magnitude of predicted y_t may increase exponentially, leading to a poorly behaved system [12]. For $\delta = 1 - ||\bar{A}||_2$, the effective memory of an SSM is $O(\frac{1}{\delta})$, as later terms will have a negligible contribution. [1]. However, as the SSM is highly non-convex in its parameters, as the matrix $\bar{A}$ is raised to large powers, it is often a challenge to fit the SSM parameters to a system. This issue is exacerbated for systems with long-term dependencies, as the SSM needs to learn a system close to instability (δ near 0) and optimization techniques may veer into instability.

The power of the SSM structures comes through impressive representational capacity, quick inference when suitable structure is imposed, parallelization training, and simple system evolution. Together, these properties make an SSM a suitable basis for designing larger, more-powerful deep models [1].

However, it is worth noting that a naive implementation will struggle with learning a system with high effective memory and have computational bottlenecks from matrix multiplication [10].

For many of the works we discuss, D is often assumed to be 0, or alternatively ignored, as D parameterizes a skip connection that could otherwise be learned by the remainder of the system.

3 Deep Sequence Models (RNNs and LSTMs)

Recently, deep sequence models have had great success in tasks ranging from translation, text prediction, image generation, and solving prediction tasks in control theory [1]. Many problems of practical importance can be phrased as sequence prediction tasks and having models capable of general input-to-output understanding paves the way for models to understand segments of the world closer to how we do. The current deep architectures for sequence to sequence prediction can be largely categorized as [6]:

- *Convolutional neural networks (CNN)*: Leverage convolutional filters to accumulate sequence information.

- *Recurrent neural networks* (RNN): Leveraging recurrent computation with a maintained state.
- *Transformers*: Leverage attention to reconstruct relevant state for each output.

Sequence to sequence models generally have limitations due to the large and variable input size. Frequently, models will struggle to capture dependencies due to having a fixed context size, optimization challenges such as a vanishing gradient, and compute limitations for long sequences [6]. Below, we introduce these three architectures, describe their major limitations, and discuss how state space models incorporate attributes of these three architectures.

3.1 Convolutional neural networks

Since the introduction of LeNet by LeCun et al. in 1998 to classify handwritten digits [17], CNNs have become the standard architecture for classifying and processing images and many other datasets with local structure. This has led to their widespread adoption in computer vision, where they remain many practitioners' tool of choice.

Historically, early convolutional neural networks (such as AlexNet, designed by Alex Krizhevsky et al. for the 2012 ImageNet competition) [16] were able to outperform competing models with hand-crafted features due to their ability to learn increasingly complex sets of features directly from the training dataset. In particular, CNNs rely on the use of a set of learned *filters*, small matrices which are convolved over their (much larger) input, to identify local regions of the input data that "match" the various filters in the corresponding layer.

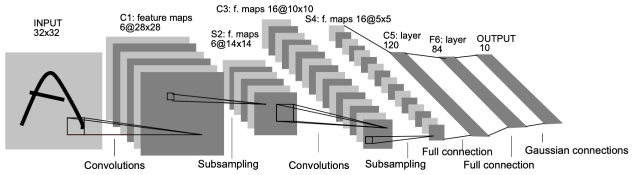


Figure 2: The architecture for LeNet, an early CNN used on the MNIST dataset. [17]

Modern CNNs will often have a *max pooling* layer after each convolutional layer to combine adjacent filter-image dot products and promote the detection of non-local features in later convolutional layers.

One major limitation of standard convolutional neural networks, however, is that their construction prevents them from applying to sequences of varying size, since the dimension of the weight matrices in the fully-connected layers at the end of the network constrains the size of each of the convolutional layers that comes before it. This prevents them from becoming applied to many instances of time-series data whose length is not predefined (such as variable-length audio recordings and natural text, among others). This issue is addressed through recurrent neural networks, discussed below, although such architectures suffer from their own share of problems (such as the vanishing gradient problem).

3.2 Recurrent neural networks

Recurrent neural networks are best suited for time-series data of variable length, where the hidden *recurrent units* on which they are based are able to process the input sequentially while maintaining a memory of past timesteps. More specifically, if $(x^{(i)})_{i=1}^{n}$ is a sequence of data, then a recurrent neural network is a feed-forward neural network defined by two relations [18]:

$$\begin{aligned} h^{(t)} &= \sigma(W_{hx}x^{(t)} + W_{hh}h^{(t-1)} + b_h) \\ y^{(t)} &= \text{softmax}(W_{yh}h^{(t)} + b_y) \end{aligned} \tag{5}$$

where $h^{(t)}$ denotes the output of hidden layer t, $y^{(t)}$ denotes the output of the RNN at time step t, and $W_{hx}, W_{hh}, W_{yh}, b_h, b_y$ are learned parameters. The defining characteristic of recurrent neural networks (as opposed to deep feed-forward neural networks) is that the trained parameters are shared between layers; the network can also be "unfolded" to form a deep neural network with t layers, as can be seen in Fig. 3 below. The hidden state can be viewed similarly as that in the SSM, storing necessary context from earlier in the input sequence although also being a bottleneck for how much information from prior sequence elements can be used in the next prediction.

Nevertheless, because the effective number of layers increases linearly with respect to the input sequence length, RNNs are especially susceptible to the *vanishing gradient* and *exploding gradient* problems, where

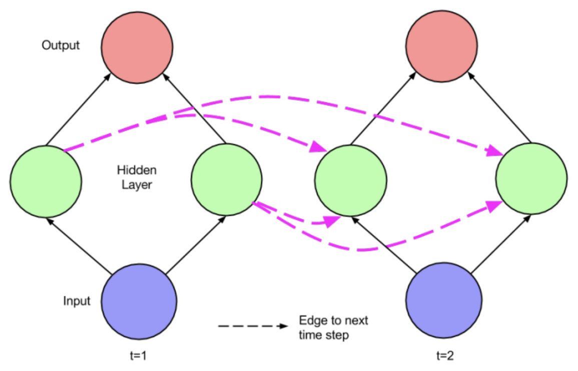


Figure 3: A simple recurrent neural network unfolded across two time steps. [18]

gradients either vanish (to zero) or explode (to infinity) as a result of repeated multiplication by small or large numbers. More specifically, for $t_0 \ll t$, the input at time t_0 passes through neuron j in each of the $t - t_0$ hidden layers with the same weights before reaching the output at time t, picking up a factor of $(W_{hh})_{jj}^{t-t_0}$ along the way. Depending on the range and behavior of activation function selected and whether $|(W_{hh})_{jj}|$ is greater than or less than 1, this can cause the resulting gradients obtained in backpropagation (with respect to the input) to either vanish or explode [18]. In particular, if the activation function σ in the hidden layers is sigmoid, the vanishing gradient problem becomes pertinent as $\sigma'(x) \leq 1$ for all x.

Various methods have been developed to prevent the vanishing gradient problem from stalling the training process. One relatively simple method for addressing the issue is to simply truncate backpropagation after a certain number of time steps, leading to truncated backpropagation through time (TBPTT). With a small "window," RNNs trained with TBPTT can avoid the vanishing/exploding gradient problems at the cost of limiting the size of the model's context window [18].

Because this is often an unacceptable tradeoff (especially for many text-based applications), Hochreiter and Schmidhuber [13] introduced *long short-term memory* (LSTMs) in 1997, which are carefully designed to avoid the vanishing gradient problem while maintaining the ability to deal with long-distance dependencies.

One other major issue with recurrent neural networks is they must be trained and evaluated sequentially due to the nontrivial dependence of $h^{(t)}$ on $h^{(t-1)}$; this issue is addressed through transformers, which use parallelizable algorithms for matrix multiplication to speed up execution.

3.3 Transformers

Unlike RNNs, transformers take advantage of self-attention to handle a larger context window. The transformer model was introduced in [22] and has now become ubiquitous with sequence-to-sequence modeling and natural language processing. We omit a full explanation of the transformer architecture, instead explaining the key improvements, and we point the reader to [14, 22, 24] for a more detailed discussion.

The transformer architecture is built on the attention layer, which allows for context accumulation between any input and all prior inputs. In generative models, the most common layer is the self-attention layer. In self-attention, for an input sequence $X \in \mathbb{R}^{L\times d}$, self-attention has learned matrices W^k, W^q, W^v, and projects X to the key, query, and value matrices defined as in Eq. 6 [24]:

$$\begin{aligned} Q &= XW^q \in \mathbb{R}^{L\times d_k} \\ K &= XW^k \in \mathbb{R}^{L\times d_k} \\ V &= XW^v \in \mathbb{R}^{L\times d_v} \end{aligned} \tag{6}$$

The (scaled dot-product) attention operation is then performed in Eq 7:

$$\text{attn}(Q,K,V) = \text{softmax}(\frac{QK^\intercal}{\sqrt{d_k}})V \in \mathbb{R}^{L\times d_v} \tag{7}$$

Alternatively, to incorporate information from another sequence, Q may the projection of a different sequence, such as in a translation task.

Note that, for the matrix $P = \text{attn}(Q,K,V)$, with query, key, and value vectors q_i, k_j, v_j (row vectors in the query, key, and value matrices) [24]:

$$P_i = \sum_{1\leq j\leq L} \text{softmax}(\frac{q_i k_j^\intercal}{\sqrt{d_k}})v_j \tag{8}$$

This can intuitively be viewed as for sequence input i, it is compared to each other input j and has scalar score $\text{softmax}(\frac{q_i k_j^\intercal}{\sqrt{d_k}})$. This scalar score corresponds to how much input i "attends" to input j, or informally how important input j is to characterizing or contextualizing

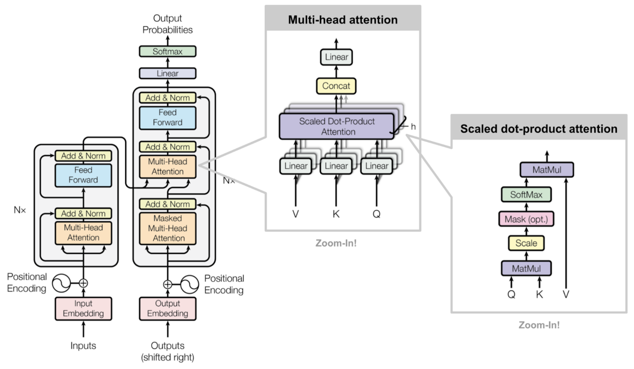


Figure 4: Vanilla transformer architecture [22, 24]

input i. We then accumulate the value vectors weighted by the attention score, to allow P_i to represent the accumulation of sequence information relevant to input i [24].

In this manner, rather than requiring a hidden state to store necessary context, the attention layer computes the relevant context, absolving the hidden state bottleneck but at the cost of computation scaling with $O(L^2)$ to predict a sequence of length L. Although modifications to attention have been proposed, the fundamental quadratic scaling in sequence length is prohibitively expensive for many applications.

To create the transformer model, the attention layer and a fully-connected layer are repeatedly layered. The full transformer architecture for a translation task is shown in Fig. 4.

The core advantages of the transformer model are easy parallelization of training and the removal of a state bottleneck, as context is recomputed for each input. However, this context computation is expensive, and State Space Models represent a return to stateful computation to avoid the runtime cost.

4 Deep sequence models as SSMs

4.1 RNN

Informed by the other sequence models in mind, RNNs can be viewed as a special case of SSMs by observing that both RNNs and SSMs attempt to approximate the same continuous-time dynamics

$$x'(t) = -x(t) + f(t, x(t)) \tag{9}$$

in similar ways [6]. Using backwards Euler discretization yields a state-dependent gated RNN, while applying Picard iteration to obtain a description for an infinitely-deep SSM [6].

In particular, Gu [6] shows through a direct application of backwards Euler discretization that single-layer discrete-time RNNs of the form

$$x_k = (1 - \sigma(z_k))x_{k-1} + \sigma(z_k)\overline{f}(k, x_{k-1}),$$

where $\overline{f}(k,x)$ is an arbitrary (discretized) function that is Lipschitz continuous in x, is actually a discretization of Eq. 9 with step sizes $\Delta_k = \exp(z_k)$, $x_k \approx x(t_k)$ with $t_k = \sum_{i=1}^{k} \Delta_i$. In some sense, then, the gating mechanism present in many RNNs can be viewed as discretizing the original dynamics of Eq. 9 with a time step Δ. [6]

Similarly, through the Picard-Lindelöf theorem, Gu shows that continuous-time infinitely-deep SSMs with oder $N = 1$ and $A = -1, B = 1, C = 1, D = 0$ also satisfy the dynamics of Eq. 9; this result can be interpreted as deeper layers approximating successive Picard iterates of the solution to the continuous-time dynamics when f is nonlinear. [6] By combining these two results, it can be concluded that infinite-depth, linear RNNs of the form

$$\begin{aligned} x_k^{(\ell)} &= (1 - \sigma(z_k))x_{k-1}^{(\ell)} + \sigma(z_k)u_k^{(\ell)} \\ u_k^{(\ell)} &= \overline{f}(k, x_k^{(\ell-1)}) \end{aligned} \tag{10}$$

also discretize the dynamics in Eq. 9. Nevertheless, since these results rely on convergence as $\ell \to \infty$, additional work still needs to be done to understand the dynamics in the non-asymptotic case (i.e., with finite depth or width) [6].

4.2 CNN

As we discussed earlier, state space models can be viewed as parameterizing sequence-length convolutions, allowing CNN properties. Various works, such as ConvSSM [21], have alternatively parameterized the SSM for differing convolutional filter properties.

As we study in Section 7.2, certain relaxations of the State Space Model, such as the Spectral Transform Unit, can be viewed equivalently as convolutional layers with fixed sequence-length filters, as shown in Fig. 8.

4.3 Attention

As we discuss in Section 6.1, although state space models cannot replicate attention layers, they can replicate linear attention layers at a dramatically reduced cost [7]. Linear attention is defined as in Eq. 11:

$$\text{attn}(Q,K,V) = (\frac{QK^\intercal}{\sqrt{d_k}})V \in \mathbb{R}^{L\times L} \tag{11}$$

By removing the softmax non-linearity, linear attention can be computed in $O(L)$ by leveraging precomputing on $K^\intercal V$ at the expense of reduced expressivity. Despite having an entirely different parameterization, [7] shows that with suitable modifications, the SSM can learn the (causal) linear attention layer.

5 Advanced State Space Models

5.1 Historical Detour: Legendre Memory Units [23]

To address the vanishing gradient problem in LSTMs, [23] propose a recurrent model structure where each layer maintains both memory and a hidden state. To store memory, a component of the Legendre Memory Unit aims to orthogonalize the continuous input $u(t) \in \mathbb{R}$ across a sliding window of length $\theta \in \mathbb{R}_{>0}$ with the ordinary differential equation:

$$\theta m'(t) = Am(t) + Bu(t)$$

With $m_t \in \mathbb{R}^d$ referring to the memory, and A, B being fixed matrices derived from the Legendre polynomials [23] . The derivation of A, B allows $m(t)$ to store the projection of the input sequence onto the Legendre polynomials, allowing $m(t)$ to approximate and efficiently store the input signal. After discretization and in its full representation, the Legendre Memory Unit approximates the above ODE, and updates its hidden state and memory with [23]:

$$\begin{aligned} h_t &= f(W_x x_t + W_h h_{t-1} + W_m m_t) \\ u_t &= e_x^T x_t + e_h^T h_{t-1} + e_m^T m_{t-1} \\ m_t &= \bar{A} m_{t-1} + \bar{B} u_t \end{aligned} \tag{12}$$

For learned kernels W_x, W_h, W_t, learned encoding vectors e_x, e_h, e_m, and a chosen nonlinearity f. Predictions can then be made based on the hidden state.

The LMU architecture vastly outperforms the LSTM in effective memory. The LMU was the first recurrent architecture capable of handling temporal dependencies across 100,000 time-steps. Success in this realm motivated the continued studying of state space dynamics,

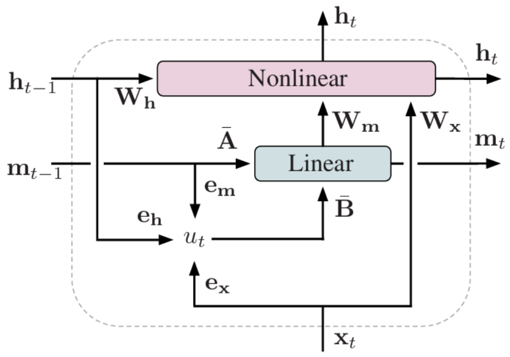


Figure 5: A time-unrolled LMU layer. [23]

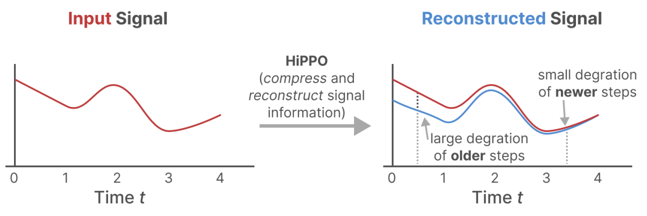
especially given that even fixed matrices A, B can lead to drastic memory improvements.

Subsequent work was greatly informed by the Legendre Memory Unit and further explored the orthogonalization of the input sequence, disregarding the vestigial LSTM components.

5.2 HiPPO: High-Order Polynomial Projection Operators [9]

Empirically, it appears that learning the SSM via gradient descent methods performs poorly in practice with random initialization of A. This is likely due to the exponential propagation, either leading to vanishing/exploding gradients. [10]

The A matrix is critical for maintaining memory, and the HIPPO approach constructs matrices that allow the hidden state to approximate the input series. Different HiPPO matrices may represent more recent signals with higher fidelity or represent with uniform fidelity. The HIPPO framework was largely inspired by the LMU memory unit [2].



Input Signal
Reconstructed Signal
HiPPO
(compress and reconstruct signal information)
small degration of newer steps
large degration of older steps
0
1
2
3
4
Time t

Figure 6: The HiPPO Approximation [20]

More specifically, HiPPO specifies a class of matrices

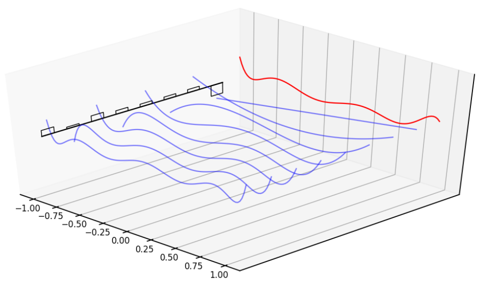


Figure 7: Decomposition of an input (red) in the Legendre basis, with the black bars representing the coefficient. The HiPPO matrix updates these coefficients at each step [20]

that allow the hidden state $x(t)$ to approximate the input history $u(t)$ in the continuous SSM as coefficients on an orthogonal basis, the most important matrix of which is the HiPPO Matrix [20].

$$\text{HiPPO Matrix:} \quad A_{nk} = \begin{cases} \sqrt{(2n+1)(2k+1)} & \text{if } n > k \\ \frac{1}{2}(2n+1) & \text{if } n = k \\ 0 & \text{if } n < k \end{cases} \tag{13}$$

Prior work has found that modifying an SSM from a random matrix A to HiPPO improves performance on the sequential MNIST classification benchmark from 60% to 98%. [20]. The HiPPO matrix maps the input sequence to its coefficients in the Legendre Polynomial basis, the set of orthogonal polynomials also used in the LMU [20]. The HiPPO matrices perform similar in the discrete case.

We can view an example of an input deconstructed into the Legendre Polynomial basis in Figure 7.

5.3 S4: Structured State Space (Sequence) Model [10]

Unfortunately, for the SSM in Eq. 2, to compute the hidden state for state dimension h and sequence length L requires $O(h^2L)$ compute and $O(hL)$ memory, which quickly becomes a computational bottleneck, requiring significantly more operations than a comparable RNN or CNN [10].

However, for diagonal A, we can easily express the exponentiation of A as:

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

And so computing $\bar{K}$ can be done theoretically in $O((h+L)\log^2(h+L))$ as a Vandermond product, although in practice this computation is numerically unstable. [10]. Note also that as (A,B,C) and $(V^{-1}AV, V^{-1}B, CV)$ parameterize the same SSM, we can learn any linear dynamical system with symmetric A with an SSM restricting A to diagonal by the spectral theorem [10].

However, despite the computational benefits of considering the class of SSMs with normal A, note that the HiPPO matrix would then not fall under this category. Motivated by the fact that the HiPPO matrix is the sum of normal and low-rank matrix (NPLR), [10] considers restricting A to be in this class as well. S4 [10] proposes an algorithm that restricts the structure of A for more efficient inference and training, with the restriction that A is NPLR still capturing an empirically valuable class of matrices. The key point is that if A is of an NPLR representation, we can compute $\bar{K}$ in $\tilde{O}(h+L)$.

The authors interleave S4 layers with linear layers and non-linear activations to form the Deep S4 Layer, which substantially outperformed transformer based approaches in the Long Range Arena benchmark [10]. Note that despite the strong performance and convenient parameterization, the S4 layer still suffers from the underlying non-convexity in the SSM fitting problem, as shown in the loss landscape in Fig. 12.

6 Princeton-Affiliated Work

6.1 Mamba [7]

Based on existing work done with state space models, in 2023 authors Tri Dao and Albert Gu introduced Mamba as an example of a *selective state space model*, constructed as a combination of "prior SSM architectures with the MLP block of Transformers" [7]. Dao and Gu identify input-dependent data selection as a key shortcoming of existing SSM models and introduce a selection mechanism by allowing the state space model parameters Δ, B, C to depend on the input. Because this prevents efficient evaluation of SSMs (because this relies on time and input invariance of the parameters), the authors develop a "hardware-aware algorithm that computes the model recurrently with a scan instead of convolution". [7]

Both Mamba and its successor, Mamba-2, impose constraints on the form of the matrix A in order to improve efficiency at the cost of decreasing model expressivity, creating *structured state space models*. In particular, Mamba-2 simplifies A by identifying $A_t = cI$ for some scalar $c \in \mathbb{R}$ (rather than Mamba's restriction of A to the diagonal matrices as in S4), which provides significant boosts to training efficiency [4], as described below.

Within their analysis, Dao and Gu introduce the idea of *state space duality* (SSD) in order to connect the notions of state space models and attention together through *N-semiseparable matrices*, lower-triangular matrices M satisfying

$$M_{ji} = C_j^{\mathsf{T}} A_j \cdots A_{i+1} B_i$$

for vectors $B_0, \ldots, B_{T-1}, C_0, \ldots, C_{T_1} \in \mathbb{R}^N$ and square matrices $A_0 \ldots, A_{T-1} \in \mathbb{R}^{N \times N}$ [4].

In particular, when A is a constant multiple of the identity matrix, the structured state space model specified by such a choice of A reduces to considering the semiseparable matrix M defined by $M_{ji} = A_{j:i} \cdot (C_j^{\mathsf{T}}, B_i)$, which can be vectorized as $M = L \circ (CB^{\mathsf{T}})$ for

$$L = \begin{pmatrix} 1 & & & & \\ a_1 & 1 & & & \\ a_2 a_1 & a_2 & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \\ a_{T-1} \cdots a_1 & a_{T-1} \cdots a_2 & \cdots & a_{T-1} & 1 \end{pmatrix}$$

[4]. Evaluation then is equivalent to calculating $y = Mx = (L \circ CB^{\mathsf{T}})x$, which is equivalent in form to causal linear attention $Y = (L \circ QK^{\mathsf{T}})V$ by associating $(L, C, B, X) \to (L, Q, K, V)$. Since linear attention does not have a softmax layer preventing us from precomputing $K^{\mathsf{T}}V$, as in the standard formulation of attention found in transformers, constraining $A = A_t I$ for $A_t \in \mathbb{R}$ means that we can reduce both the size and improve the efficiency of the structured state space model, and the L matrix corresponds to "input-dependent relative positional encodings" that allows for Mamba's "selectivity" [8]. Note that L also prevents query-key combinations where the key is further along in the sequence than the query, ensuring the "causal" nature.

6.2 Spectral Transform Unit [12]

Similar to S4, the Hazan Lab, led by Prof. Elad Hazan, and Google Deepmind [1, 5, 12, 19] explored structured state space models, restricting that the matrix A is diagonal. As before, as (A, B, C) and $(V^{-1}AV, V^{-1}B, CV)$ parameterize the same LDS ($D = 0$), we have that restricting A diagonal still allows the state space model to learn any LDS with symmetric A. [10]

Note that for diagonal A and assuming $D = 0$ as typical, we can continue from Eq. 4. Notating A as a diagonal matrix with diagonals $\lambda_1, \lambda_2, \ldots, \lambda_K$, B having row vectors indexed b_j, and C having column vectors c_j [12]:

$$\begin{aligned} y_t &= \sum_{i=1}^{t-1} \bar{C} A^i \bar{B} u_{t-i} \\ &= \sum_{i=1}^{t-1} \sum_{j=1}^{k} c_j \lambda_j^i b_j u_{t-i} \\ &= \sum_{i=1}^{t-1} \sum_{j=1}^{k} c_j b_j \lambda_j^i u_{t-i} \\ &= \sum_{j=1}^{k} c_j b_j \sum_{i=1}^{t-1} \lambda_j^i u_{t-i} \\ &= \sum_{j=1}^{k} c_j b_j \sum_{i=1}^{t-1} \mu(\lambda_j)(i) u_{t-i} \end{aligned} \tag{14}$$

Where, note that b_j is a row vector so $c_j b_j$ is a matrix, and where μ is the function with $\mu(\alpha) = (1, \alpha, \alpha^2, \ldots)$. We still have a dependence on the hidden dimension k through the λ_j, but [12] finds this to be removable.

Remarkably, for $0 \leq \alpha \leq 1$, there is a basis that can represent any vector $\mu(\alpha)$ with exponentially decaying error in the amount of basis vectors [12]. Thus, practically speaking, with 20 basis vectors $\phi_1, \ldots, \phi_{20}$, we can represent any $\mu(\alpha)$ (or linear combination of $\mu(\lambda_i), \ldots, \mu(\lambda_k)$) with low error. Thus, there exists a matrix M and coefficients β_i, such that:

$$\sum_{j=1}^{k} c_j b_j \sum_{i=1}^{t-1} \mu(\lambda_j) = M \sum_{i=1}^{20} \beta_i \phi_i$$

Thus,

$$y_t = M \sum_{i=1}^{20} \beta_i \phi_i * u$$

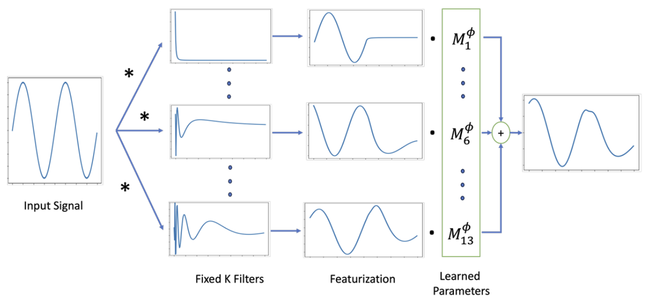


Figure 8: Viewpoint of the spectral parametrization as learning coefficients on fixed filters [12]

Moreover, we can wrap the coefficients β_i in the matrix M, and so for some M, we have:

$$y_t = M \sum_{i=1}^{20} \phi_i * u$$

This is an incredibly convenient parametrization of the State Space Model and removes the extreme non-convexity inherent in traditional approaches to fitting λ_i, as λ_i is typically raised to high powers in the sequence, whereas this approach bypasses learning λ_i by learning a convention basis representation. Additionally, as the size of M is not dependent on k, this approach can fit any linear dynamical system with parameters independent of the hidden dimension size.

The model unit detailed, and shown in Fig. 8, is known as the Spectral Transform Unit. It is also worth noting that the Spectral Transform Unit parametrizes a larger class of function than just linear dynamical systems with symmetric A, as the filters can recombine into terms not expressive as combinations of $\mu(\alpha)$.

Much of the power of this approach is through the basis ϕ, also known as spectral filters, of the functions $\mu(\alpha)$, which transforms a highly non-convex learning problem to fitting coefficients of a basis. Hazan et. al [12] compute the filters as the top eigenvalues of the matrix:

$$Z = \int_0^1 \mu(\alpha) \otimes \mu(\alpha) d\alpha$$

We can visualize the these filters in Fig. 9. Matrix Z, as a Hankel Matrix, has exponential eigenvalue decay, which allows for accurate low-rank approximation, enabling the use of a few basis elements to accurate model the function class $\mu(\alpha)$ [12].

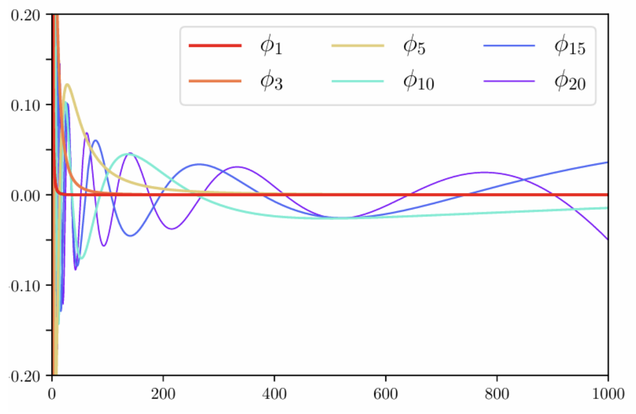


Figure 9: The entries of some of the spectral filters. The x-axis is the time domain [12]

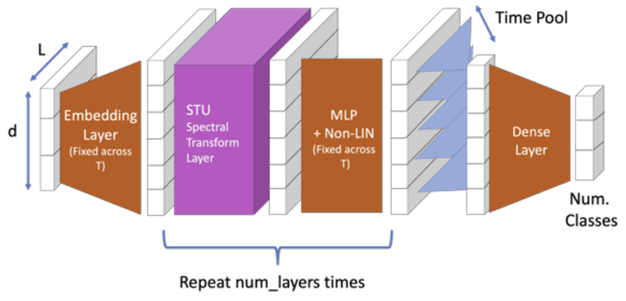
Hazan et. al. combine the Spectral Transform Unit (STU), with an autoregressive component to model the system with:

$$y_t = y_{t-2} + \sum_{i=1}^{3} M_i' u_{t+1-i} + M \sum_{i=1}^{20} \phi_i * u$$

Additionally, to model negative eigenvalues, Hazen. et. al [1] introduce additional filters that allow the representation of $\mu(\alpha)$ for $\alpha < 0$.

6.3 Spectral Transformer [19]

The transformer model consists of nesting attention and fully-connected layers. Hazan. et. al. leverage a similar architecture, interleaving STU and deep layers for greater expressivity into the Stacked STU, as shown in Fig. 10 [1].



L
d
Embedding Layer (Fixed across T)
STU Spectral Transform Layer
MLP + Non-LIN (Fixed across T)
Time Pool
Dense Layer
Num. Classes
Repeat num_layers times

Figure 10: Multi-layer Stacked STU Model [1]

The Stacked STU has had remarkable performance on the Long Range Area benchmark [1], greatly surpassing transformer models and other State Space Models on most benchmarks. Notably, no transformer model has performed beyond random chance at the Path-X image sequence benchmark, while the Stacked STU achieves 93.24 accuracy (although underperforming other SSM models). Additionally, the Spectral Transform Unit has extremely stable training, being robust to parameter changes unlike prior State Space Models (such as S4) [1].

Subsequent work [19] from the Hazan Lab has dramatically scaled up the Stacked STU model and engaged in more detailed analysis. Notably, experimentation has confirmed that the loss landscape of the STU is fairly smooth as hypothesized, as shown in Fig. 12.

The larger 2.6 billion parameter Flash STU model,[1] which has an architecture comprising of STU layers and (sliding window) attention, as shown in Fig. 11, outperforms transformer models of the same scale on language modeling tasks, with more stable training and fewer loss spikes [19].

7 Remarks

With the introduction of the State Space Model, it seems that the current transformer-dominant paradigm may be poised to shift. Startups such as Cartesia[2] are using state space models as the basis for multi-modal intelligence, and research such as Mamba and the STU [1, 7] show that SSMs can substantially exceed transformer performance on long range tasks when their expressivity is carefully constrained. The Flash STU model has shown that SSMs may be able to outperform the transformer model on language and can also match the speed of high-optimized transformer implementations [19]. Highly structured SSMs (for example, when A takes on a scalar-times-identity form) are expressible in terms of causal linear attention, further showing how SSMs serve as a generalization of the attention mechanism and the convolution.

While it is unclear so far whether state space models will maintain their advantage with larger and larger model sizes, the existing work is promising. SSMs have

[1]Implementation available at https://github.com/windsornguyen/flash-stu/

[2]https://www.cartesia.ai/

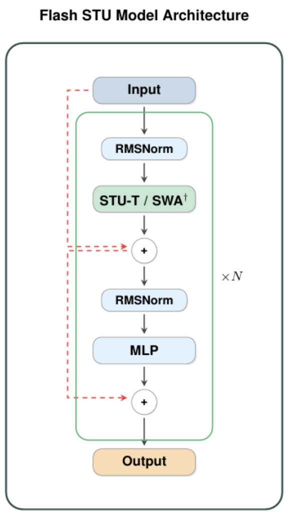


Figure 11: Flash STU architecture [19]

continued to shine at the largest scales they have been trained at, match transformer speed without yet being fully optimized, and the stability of training of models such Flash STU (2.6B) [19] is remarkable. We are excited to have presented recent developments in state space models and look forward to their continued success.

References

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

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

[3] Dao, T. *Hardware-Aware Algorithms for Efficient Machine Learning*. Phd dissertation, Stanford University, June 2023.

[4] Dao, T., and Gu, A. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality, 2024.

[5] Google DeepMind, P. Spectral transformers. https://sites.google.com/view/gbrainprinceton/projects/spectral-transformers, 2024.

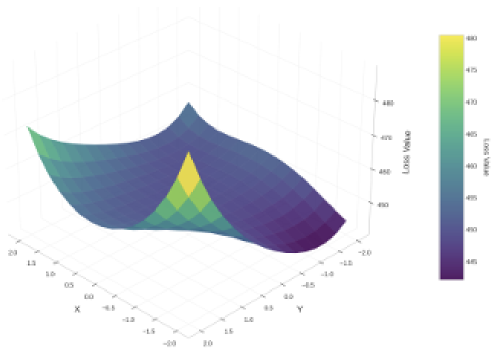


Figure 3: Local loss landscape of the **STU** layer.

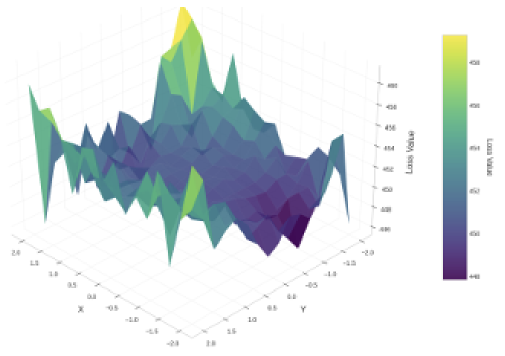


Figure 4: Local loss landscape of the **S4** layer.

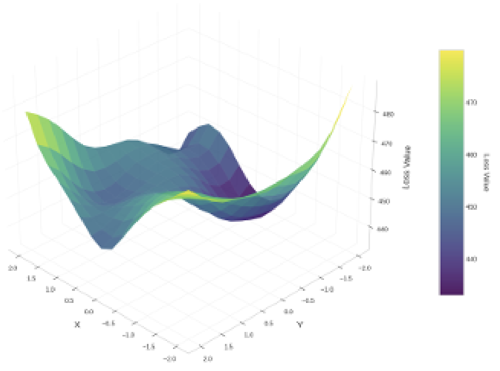


Figure 5: Local loss landscape of the **Mamba-2** layer.

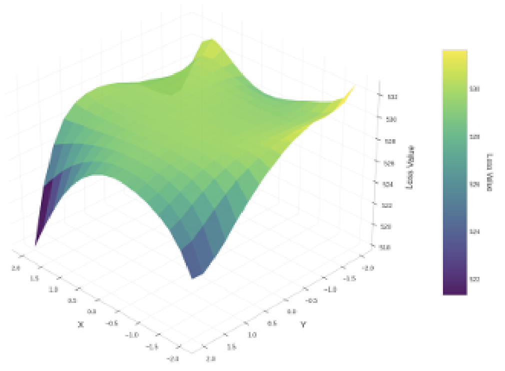


Figure 6: Local loss landscape of the **attention** layer.

Figure 12: Loss landscapes of various approaches to fitting linear dynamical systems [19]

[6] Gu, A. *Modeling Sequences with Structured State Spaces*. PhD thesis, Stanford University, Stanford, CA, June 2023.

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

[8] Gu, A., and Dao, T. State space duality (mamba-2) part i - the model. https://tridao.me/blog/2024/mamba2-part1-model, 2024.

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

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

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

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

[13] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural Computation 9*, 8 (1997), 1735–1780.

[14] Huang, A., Subramanian, S., Sum, J., Almubarak, K., Biderman, S., and Rush, S. The annotated transformer. https://nlp.seas.harvard.edu/annotated-transformer/, 2022.

[15] 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.

[16] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet classification with deep convolutional neural networks. *Commun. ACM 60*, 6 (May 2017), 84–90.

[17] LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[18] LIPTON, Z. C., BERKOWITZ, J., AND ELKAN, C. A critical review of recurrent neural networks for sequence learning, 2015.

[19] LIU, Y. I., NGUYEN, W., DEVRE, Y., DOGARIU, E., MAJUMDAR, A., AND HAZAN, E. Flash stu: Fast spectral transform units, 2024.

[20] RUSH, S., AND KARAMCHETI, S. The annotated s4: Efficiently modeling long sequences with structured state spaces. `https://srush.github.io/annotated-s4/`, 2024.

[21] SMITH, J. T. H., MELLO, S. D., KAUTZ, J., LINDERMAN, S. W., AND BYEON, W. Convolutional state space models for long-range spatiotemporal modeling, 2023.

[22] VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, L., AND POLOSUKHIN, I. Attention is all you need, 2023.

[23] VOELKER, A. R., KAJIĆ, I., AND ELIASMITH, C. Legendre memory units: Continuous-time representation in recurrent neural networks. In *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019)* (Vancouver, Canada, 2019).

[24] WENG, L. Attention? attention! *lilianweng.github.io* (2018).