Introduction to State Space Models (SSM)

Community Article Published July 19, 2024

Loïck BOURDOIS
lbourdois Follow

Une version en français est disponible sur mon blog.

> **Changelog**
> 2023-12-14: article release.
> 2024-04-08: typos corrections (my English isn't perfect 😅).
> 2024-06-11: added links to the second article of my SSM blog posts serie.
> 2024-07-18: LaTex typo correction.
> 2024-09-23: rewrote introduction and added a section about the origin of SSM in deep learning.
> 2025-10-26: clarification of some sentence structures following reader feedback.
> 2025-11-01: add some ressources (Appendix E of the book *Hands-On Machine Learning with Scikit-Learn and Pytorch* by Aurélien Geron)

Foreword

I'd like to extend my warmest thanks to Boris ALBAR, Pierre BEDU and Nicolas PREVOT for agreeing to set up a working group on the subject of SSMs and thus

accompanying me in my discovery of this type of model. A special thanks to the former for taking the time to proofread this blog post.

Introduction

The ***States Spaces Models*** are traditionally used in control theory to model a dynamic system via state variables.

Aaron R. VOELKER and Chris ELIASMITH addressed the question of how the brain effectively represents temporal information. They discovered in 2018 in "Improving Spiking Dynamical Networks: Accurate Delays, Higher-Order Synapses, and Time Cells" that an SSM is an excellent model for describing the "time cells" present in the brain (hippocampus and cortex in particular).

From neuroscience, they applied their work to the field of deep learning and were thus (to our knowledge) the first to use SSMs in deep learning. For more details on this work, please refer to the "SSM history" section at the end of this blog post.

In this article, we will define the basics of a deep learning SSM. To do this, we will based on the S4 model introduced in "*Efficiently Modeling Long Sequences with Structured State Spaces*" by Albert GU et al. in 2021. This is not a model that is used as is in practice (other SSMs with better performance or easier to implement are now available). We use it here for educational purposes. Released a week earlier than S4, LSSL, by the same authors, is also an important source of information on the subject. We'll take a look at the various developments arising from S4 in a future blog post. Before that, let's delve into the basics of SSM.

Definition of an SSM in deep learning

Let's use the image below to define an SSM:

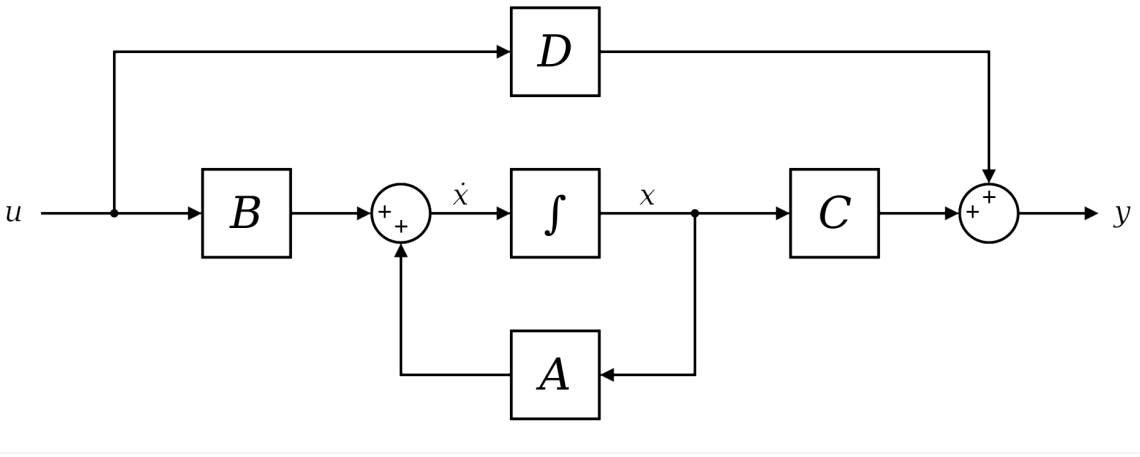


Figure 1: *View of a continuous, time-invariant SSM (Source: https://en.wikipedia.org/wiki/State-space_representation)*

It can be seen that an SSM is based on three variables that depend on time t :

- $x(t) \in \mathbb{C}^n$ represents the n state variables,
- $u(t) \in \mathbb{C}^m$ represents the m state inputs,
- $y(t) \in \mathbb{C}^p$ represents the p outputs,

We can also see that it's made up of four learnable matrices: $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$.

- $\mathbf{A} \in \mathbb{C}^{n \times n}$ is the state matrix (controlling the latent state x),
- $\mathbf{B} \in \mathbb{C}^{n \times m}$ is the control matrix,
- $\mathbf{C} \in \mathbb{C}^{p \times n}$ is the output matrix,
- $\mathbf{D} \in \mathbb{C}^{p \times m}$ is the command matrix,

The above picture can be reduced to the following system of equations:

$$x'(t) = \mathbf{A}x(t) + \mathbf{B}u(t)$$
$$y(t) = \mathbf{C}x(t) + \mathbf{D}u(t)$$

Note: here we use the notation x' to designate the derivative of x. It's not out of the question to encounter the notation $\dot{x}$ in the literature instead.

Similarly, since it is implicit that the variables depend on time, the preceding equation is generally written in the following form for the sake of simplicity:

$$x' = \mathbf{A}x + \mathbf{B}u$$
$$y = \mathbf{C}x + \mathbf{D}u$$

This system can be made even lighter, because in deep learning SSMs, $\mathbf{D}u = 0$ is seen as an easily computable *skip connection*.

$$x' = \mathbf{A}x + \mathbf{B}u$$
$$y = \mathbf{C}x$$

This system is continuous. It must therefore first be discretized before it can be supplied to a computer.

Discretization

Discretization is one of, if not the most important point in SSM. All the efficiency of this architecture lies in this step, since it enables us to pass from the continuous view of the SSM to its two other views: the **recursive view** and the **convolutive view**.
If there's one thing to remember from this article, it's this.

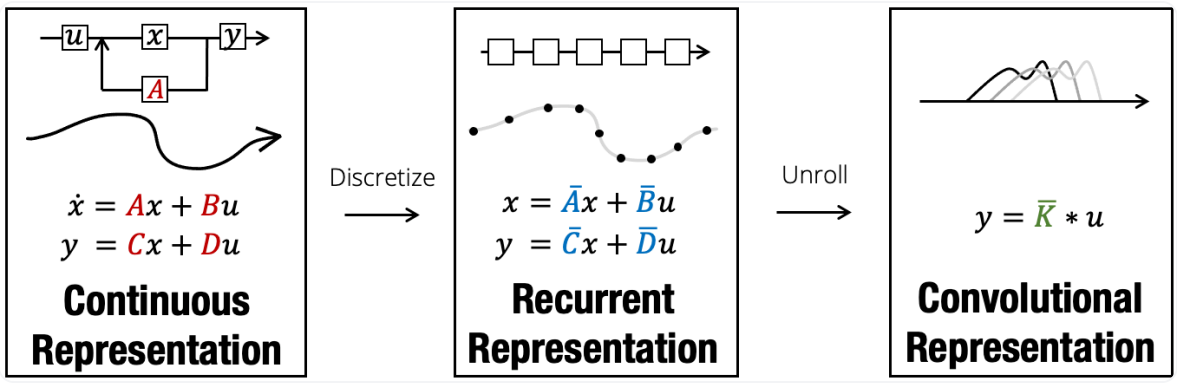


Figure 2: *Image from blog post Structured State Spaces: Combining Continuous-Time, Recurrent, and Convolutional Models by Albert GU et al. (2022)*

We'll see in later article that there are several possible discretizations. This is one of the main differences between the various existing SSM architectures.
For this first article, let's apply the discretization proposed in S4 to illustrate the two additional views of an SSM.

Recursive view of an SSM

To discretize the continuous case, let's use the trapezoid method where the principle is to assimilate the region under the representative curve of a function f defined on a segment $[t_n, t_{n+1}]$ to a trapezoid and calculate its area T : $T = (t_{n+1} - t_n)\frac{f(t_n)+f(t_{n+1})}{2}$.

We then have: $x_{n+1} - x_n = \frac{1}{2}\Delta(f(t_n) + f(t_{n+1}))$ with $\Delta = t_{n+1} - t_n$.
If $x'_n = \mathbf{A}x_n + \mathbf{B}u_n$ (first line of the SSM equation), corresponds to f, so:

$$x_{n+1} = x_n + \frac{\Delta}{2}(\mathbf{A}x_n + \mathbf{B}u_n + \mathbf{A}x_{n+1} + \mathbf{B}u_{n+1})$$
$$\iff x_{n+1} - \frac{\Delta}{2}\mathbf{A}x_{n+1} = x_n + \frac{\Delta}{2}\mathbf{A}x_n + \frac{\Delta}{2}\mathbf{B}(u_{n+1} + u_n)$$
$$(*) \iff (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})x_{n+1} = (\mathbf{I} + \frac{\Delta}{2}\mathbf{A})x_n + \Delta\mathbf{B}u_{n+1}$$
$$\iff x_{n+1} = (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})^{-1}(\mathbf{I} + \frac{\Delta}{2}\mathbf{A})x_n + (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})^{-1}\Delta\mathbf{I}$$

(*) $u_{n+1} \overset{\Delta}{\simeq} u_n$ (the control vector is assumed to be constant over a small Δ).

We've just obtained our discretized SSM!
To make this completely explicit, let's pose:

$$\bar{\mathbf{A}} = (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})^{-1}(\mathbf{I} + \frac{\Delta}{2}\mathbf{A})$$
$$\bar{\mathbf{B}} = (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})^{-1}\Delta\mathbf{B}$$
$$\bar{\mathbf{C}} = \mathbf{C}$$

We then have

$$x_k = \bar{\mathbf{A}}x_{k-1} + \bar{\mathbf{B}}u_k$$
$$y_k = \bar{\mathbf{C}}x_k$$

The notation of matrices with a bar was introduced in S4 to designate matrices in the discrete case and has since become a convention in the field of SSM applied to deep learning.

Convolutive view of an SSM

This recurrence can be written as a convolution. To do this, simply iterate the equations of the system

$$x_k = \bar{\mathbf{A}}x_{k-1} + \bar{\mathbf{B}}u_k$$
$$y_k = \bar{\mathbf{C}}x_k$$

Let's start with the first line of the system:
Step 0: $x_0 = \bar{\mathbf{B}}u_0$
Step 1: $x_1 = \bar{\mathbf{A}}x_0 + \bar{\mathbf{B}}u_1 = \bar{\mathbf{A}}\bar{\mathbf{B}}u_0 + \bar{\mathbf{B}}u_1$
Step 2: $x_2 = \bar{\mathbf{A}}x_1 + \bar{\mathbf{B}}u_2 = \bar{\mathbf{A}}(\bar{\mathbf{A}}\bar{\mathbf{B}}u_0 + \bar{\mathbf{B}}u_1) + \bar{\mathbf{B}}u_2 = \bar{\mathbf{A}}^2\bar{\mathbf{B}}u_0 + \bar{\mathbf{A}}\bar{\mathbf{B}}u_1 + \bar{\mathbf{B}}u_2$
We have x_k which can be written as a function f parametrized by $(u_0, u_1, ...u_k)$.

Let's move on to the second line of the system, where we can now inject the x_k values calculated just now:
Step 0: $y_0 = \bar{\mathbf{C}}x_0 = \bar{\mathbf{C}}\bar{\mathbf{B}}u_0$
Step 1: $y_1 = \bar{\mathbf{C}}x_1 = \bar{\mathbf{C}}(\bar{\mathbf{A}}\bar{\mathbf{B}}u_0 + \bar{\mathbf{B}}u_1) = \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}}u_0 + \bar{\mathbf{C}}\bar{\mathbf{B}}u_1$
Step 2: $y_2 = \bar{\mathbf{C}}x_2 = \bar{\mathbf{C}}(\bar{\mathbf{A}}^2\bar{\mathbf{B}}u_0 + \bar{\mathbf{A}}\bar{\mathbf{B}}u_1 + \bar{\mathbf{B}}u_2) = \bar{\mathbf{C}}\bar{\mathbf{A}}^2\bar{\mathbf{B}}u_0 + \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}}u_1 + \bar{\mathbf{C}}\bar{\mathbf{B}}u_2$
We can observe the convolution kernel $\bar{\mathbf{K}}_k = (\bar{\mathbf{C}}\bar{\mathbf{B}}, \bar{\mathbf{C}}\bar{\mathbf{A}}\bar{\mathbf{B}}, ..., \bar{\mathbf{C}}\bar{\mathbf{A}}^k\bar{\mathbf{B}})$ applicable to u_k, hence $K * u$.

As with matrices, we apply a bar to the $\bar{\mathbf{K}}$ to specify that it is the convolution kernel obtained after discretization. It is generally referred to as the **SSM convolution kernel** in the literature, and its size is equivalent to the entire input sequence. This convolution kernel is calculated by Fast Fourier Transform (FFT) and will be explained in future articles.

Advantages and limitations of each of the three views

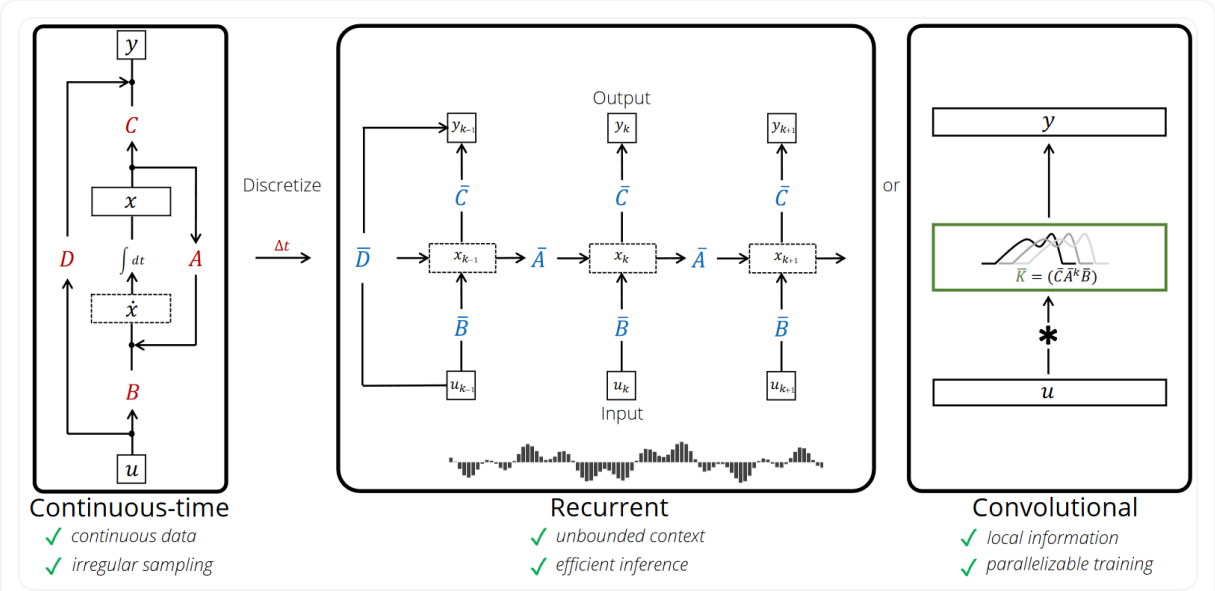


Figure 3: *Image from the paper Combining Recurrent, Convolutional, and Continuous-time Models with Linear State-Space Layers by Albert GU et al, released a week before S4*

The different views of SSM each have their advantages and disadvantages - let's take a closer look.

For the **continuous view**, the advantages and disadvantages are as follows:
✓ Automatically handles continuous data (audio signals, time series, for example). This represents a huge practical advantage when processing data with irregular or time-shifted sampling.
✓ Mathematically feasible analysis, e.g. by calculating exact trajectories or building memory systems (HiPPO).
✗ Extremely slow for both training and inference.

For the **recursive view** these are the well-known advantages and disadvantages of recursive neural networks, namely:
✓ Natural inductive bias for sequential data, and in principle unbounded context.
✓ Efficient inference (constant-time state updates).
✗ Slow learning (lack of parallelism).
✗ Gradient disappearance or explosion when training too-long sequences.

For the **convolutional view**, we're talking here about the well-known advantages and disadvantages of convolutional neural networks (we're here in the context of their one-dimensional version), namely:

✓ Local, interpretable features.

✓ Efficient (parallelizable) training.

✗ Slowness in online or autoregressive contexts (must recalculate entire input for each new data point).

✗ Fixed context size.

So, depending on the stage of the process (training or inference) or the type of data at our disposal, it is possible to switch from one view to another in order to fall back on a favorable framework for getting the most out of the model.
We prefer the convolutional training view for fast training via parallelization, the recursive view for efficient inference, and the continuous view for handling continuous data.

Learning matrices

In the convolution kernel developed above, $\bar{\mathbf{C}}$ (a row vector) and $\bar{\mathbf{B}}$ (a column vector), are learnable.
Concerning $\bar{\mathbf{A}}$, we've seen that in our convolution kernel, it's expressed as a power of k at time k. This can be very time-consuming to calculate, so we're looking for a fixed $\bar{\mathbf{A}}$. For this, the best option is to have it diagonal:

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

By the spectral theorem of linear algebra, this is exactly the class of normal matrices.

In addition to the choice of discretization mentioned above, the way in which $\bar{\mathbf{A}}$ is defined and initiated is one of the points that differentiates the various SSM architectures developed in the literature, which we'll develop in the next blog post. Indeed, empirically, it appears that an SSM initialized with a random $\mathbf{A}$ matrix leads to poor results, whereas an initialization based on the **HiPPO** matrix (for *High-Order Polynomial Projection Operator*) gives very good results (from 60% to 98% on the MNIST sequential benchmark).

The **HiPPO** matrix was introduced by the S4 authors in a previous paper (2020). It is included in the LSSL paper (2021), also by the S4 authors, as well as in the S4 appendix. Its formula is as follows:

$$\mathbf{A} = \begin{bmatrix} 1 & & & & & & & & \\ -1 & 2 & & & & & & & \\ 1 & -3 & 3 & & & & & & \\ -1 & 3 & -5 & 4 & & & & & \\ 1 & -3 & 5 & -7 & 5 & & & & \\ -1 & 3 & -5 & 7 & -9 & 6 & & & \\ 1 & -3 & 5 & -7 & 9 & -11 & 7 & & \\ -1 & 3 & -5 & 7 & -9 & 11 & -13 & 8 & \\ \vdots & & & & & & & & \ddots \end{bmatrix}$$

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

(Note: here is the HiPPO-LegT version, check out this section of the following blog post to learn more about the different existing forms.).

This matrix is not normal, but it can be decomposed as a normal matrix plus a matrix of lower rank (summarized in the paper as NPLR for *Normal Plus Low Rank*). The authors prove in their paper that this type of matrix (and especially their power) can be computed efficiently via three techniques (see Algorithm 1 in the paper): truncated generating series, Cauchy kernels and Woodbury identity.

Details of the demonstration showing that an NPLR matrix can be computed efficiently as a diagonal matrix can be found in the appendix (see part B and C) of the paper LSSL.

The authors of S4 subsequently made modifications to the **HiPPO** matrix (on how to initiate it) in their paper *How to Train Your HiPPO* (2022). The model resulting from this paper is generally referred to as "S4 V2" or "S4 updated" in the literature as opposed to the "original S4" or "S4 V1".

In the next article, we'll see that other authors (notably Ankit GUPTA) have proposed using a diagonal matrix instead of an NPRL matrix, an approach that is now preferred as it is simpler to implement.

Experimental results

Let's end this blog post by analyzing a selection of the S4's results on various tasks and benchmarks to get a feel for the potential of SSMs.

Let's start with an audio task and the benchmark *Speech Commands* by WARDEN (2018).

Table 11: (**Speech Commands classification.**) Test accuracy on 35-way keyword spotting. Training examples are 1-second audio waveforms sampled at 16000Hz, or a 1-D sequence of length 16000. Last column indicates 0-shot testing at 8000Hz where examples are constructed by naive decimation.

Model	Parameters	16000Hz	8000Hz
S4-LegS	307K	96.08 (0.15)	91.32 (0.17)
S4-FouT	307K	95.27 (0.20)	91.59 (0.23)
S4-(LegS/FouT)	307K	95.32 (0.10)	90.72 (0.68)
InceptionNet	481K	61.24 (0.69)	05.18 (0.07)
ResNet-18	216K	77.86 (0.24)	08.74 (0.57)
XResNet-50	904K	83.01 (0.48)	07.72 (0.39)
ConvNet	26.2M	95.51 (0.18)	07.26 (0.79)

Figure 4: Image from the paper On the Parameterization and Initialization of Diagonal State Space Models by Albert GU et al. (2022), also known as S4D, published after S4 but which

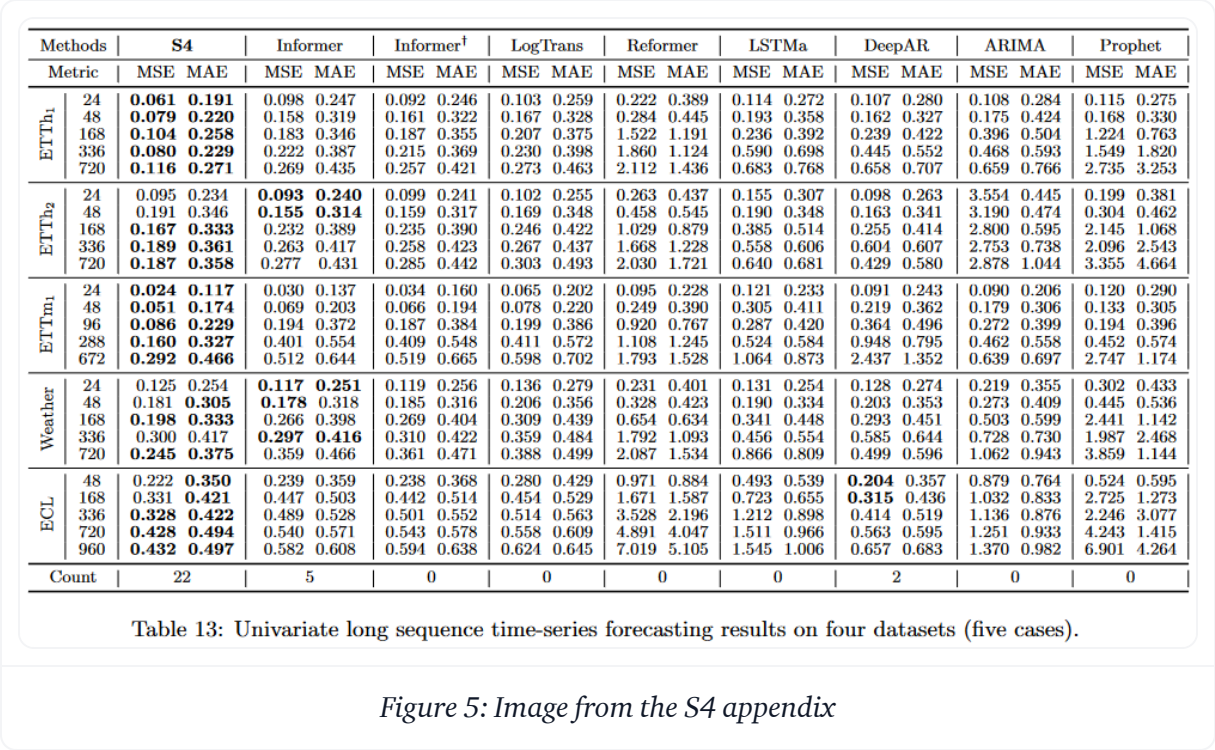
Table 11: (**Speech Commands classification.**) Test accuracy on 35-way keyword spotting. Training examples are 1-second audio waveforms sampled at 16000Hz, or a 1-D sequence of length 16000. Last column indicates 0-shot testing at 8000Hz where examples are constructed by naive decimation.

Model	Parameters	16000Hz	8000Hz
S4-LegS	307K	96.08 (0.15)	91.32 (0.17)
S4-FouT	307K	95.27 (0.20)	91.59 (0.23)
S4-(LegS/FouT)	307K	95.32 (0.10)	90.72 (0.68)
InceptionNet	481K	61.24 (0.69)	05.18 (0.07)
ResNet-18	216K	77.86 (0.24)	08.74 (0.57)
XResNet-50	904K	83.01 (0.48)	07.72 (0.39)
ConvNet	26.2M	95.51 (0.18)	07.26 (0.79)

reproduces in a more structured form the results of S4 for this benchmark (the results of S4D having been removed from the image so as not to spoil the next article ;)

Several things can be observed in this table.
Firstly, for a more or less equivalent number of parameters, the S4 performs much better (at least +13%) than the other models, here of the ConvNet type.
Secondly, to achieve equivalent performance, a ConvNet requires 85 times more parameters.
Thirdly, a ConvNet trained on 16K Hz gives very poor results when then applied to 8K Hz data. In contrast, the S4 retains 95% of its performance on this resampling. This can be explained by the continuous view of the SSM, where it was sufficient to halve the Δ value at the time of the test phase.

Let's continue with a time series task (introduced in a revision of S4).



Methods		S4		Informer		Informer†		LogTrans		Reformer		LSTMa		DeepAR		ARIMA		Prophet	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
$ETTh_1$	24	**0.061**	**0.191**	0.098	0.247	0.092	0.246	0.103	0.259	0.222	0.389	0.114	0.272	0.107	0.280	0.108	0.284	0.115	0.275
	48	**0.079**	**0.220**	0.158	0.319	0.161	0.322	0.167	0.328	0.284	0.445	0.193	0.358	0.162	0.327	0.175	0.424	0.168	0.330
	168	**0.104**	**0.258**	0.183	0.346	0.187	0.355	0.207	0.375	1.522	1.191	0.236	0.392	0.239	0.422	0.396	0.504	1.224	0.763
	336	**0.080**	**0.229**	0.222	0.387	0.215	0.369	0.230	0.398	1.860	1.124	0.590	0.698	0.445	0.552	0.468	0.593	1.549	1.820
	720	**0.116**	**0.271**	0.269	0.435	0.257	0.421	0.273	0.463	2.112	1.436	0.683	0.768	0.658	0.707	0.659	0.766	2.735	3.253
$ETTh_2$	24	0.095	0.234	**0.093**	**0.240**	0.099	0.241	0.102	0.255	0.263	0.437	0.155	0.307	0.098	0.263	3.554	0.445	0.199	0.381
	48	0.191	0.346	**0.155**	**0.314**	0.159	0.317	0.169	0.348	0.458	0.545	0.190	0.348	0.163	0.341	3.190	0.474	0.304	0.462
	168	**0.167**	**0.333**	0.232	0.389	0.235	0.390	0.246	0.422	1.029	0.879	0.385	0.514	0.255	0.414	2.800	0.595	2.145	1.068
	336	**0.189**	**0.361**	0.263	0.417	0.258	0.423	0.267	0.437	1.668	1.228	0.558	0.606	0.604	0.607	2.753	0.738	2.096	2.543
	720	**0.187**	**0.358**	0.277	0.431	0.285	0.442	0.303	0.493	2.030	1.721	0.640	0.681	0.429	0.580	2.878	1.044	3.355	4.664
$ETTm_1$	24	**0.024**	**0.117**	0.030	0.137	0.034	0.160	0.065	0.202	0.095	0.228	0.121	0.233	0.091	0.243	0.090	0.206	0.120	0.290
	48	**0.051**	**0.174**	0.069	0.203	0.066	0.194	0.078	0.220	0.249	0.390	0.305	0.411	0.219	0.362	0.179	0.306	0.133	0.305
	96	**0.086**	**0.229**	0.194	0.372	0.187	0.384	0.199	0.386	0.920	0.767	0.287	0.420	0.364	0.496	0.272	0.399	0.194	0.396
	288	**0.160**	**0.327**	0.401	0.554	0.409	0.548	0.411	0.572	1.108	1.245	0.524	0.584	0.948	0.795	0.462	0.558	0.452	0.574
	672	**0.292**	**0.466**	0.512	0.644	0.519	0.665	0.598	0.702	1.793	1.528	1.064	0.873	2.437	1.352	0.639	0.697	2.747	1.174
Weather	24	0.125	0.254	**0.117**	**0.251**	0.119	0.256	0.136	0.279	0.231	0.401	0.131	0.254	0.128	0.274	0.219	0.355	0.302	0.433
	48	0.181	**0.305**	**0.178**	0.318	0.185	0.316	0.206	0.356	0.328	0.423	0.190	0.334	0.203	0.353	0.273	0.409	0.445	0.536
	168	**0.198**	**0.333**	0.266	0.398	0.269	0.404	0.309	0.439	0.654	0.634	0.341	0.448	0.293	0.451	0.503	0.599	2.441	1.142
	336	0.300	0.417	**0.297**	**0.416**	0.310	0.422	0.359	0.484	1.792	1.093	0.456	0.554	0.585	0.644	0.728	0.730	1.987	2.468
	720	**0.245**	**0.375**	0.359	0.466	0.361	0.471	0.388	0.499	2.087	1.534	0.866	0.809	0.499	0.596	1.062	0.943	3.859	1.144
ECL	48	0.222	**0.350**	0.239	0.359	0.238	0.368	0.280	0.429	0.971	0.884	0.493	0.539	**0.204**	0.357	0.879	0.764	0.524	0.595
	168	0.331	**0.421**	0.447	0.503	0.442	0.514	0.454	0.529	1.671	1.587	0.723	0.655	**0.315**	0.436	1.032	0.833	2.725	1.273
	336	**0.328**	**0.422**	0.489	0.528	0.501	0.552	0.514	0.563	3.528	2.196	1.212	0.898	0.414	0.519	1.136	0.876	2.246	3.077
	720	**0.428**	**0.494**	0.540	0.571	0.543	0.578	0.558	0.609	4.891	4.047	1.511	0.966	0.563	0.595	1.251	0.933	4.243	1.415
	960	**0.432**	**0.497**	0.582	0.608	0.594	0.638	0.624	0.645	7.019	5.105	1.545	1.006	0.657	0.683	1.370	0.982	6.901	4.264
Count		22		5		0		0		0		0		2		0		0	

Table 13: Univariate long sequence time-series forecasting results on four datasets (five cases).

Figure 5: Image from the S4 appendix

The authors of the paper take up the methodology of the Informer model by ZHOU et al. (2020) and show that their model outperforms this *transformer* on 40 of the 50 configurations. The results in the table are shown in a univariate framework, but the same is observable for a multivariate framework (table 14 in the appendix).

Let's continue with a vision task and the benchmark *sCIFAR-10* by KRIZHESKY (2009).

Table 11: **(Pixel-level image classification.)** Citations refer to the original model; additional citation indicates work from which this baseline is reported.

Model	sMNIST	pMNIST	sCIFAR
Transformer [39, 41]	98.9	97.9	62.2
CKConv [33]	99.32	98.54	63.74
TrellisNet [4]	99.20	98.13	73.42
TCN [3]	99.0	97.2	-
LSTM [16, 18]	98.9	95.11	63.01
r-LSTM [39]	98.4	95.2	72.2
Dilated GRU [5]	99.0	94.6	-
Dilated RNN [5]	98.0	96.1	-
IndRNN [22]	99.0	96.0	-
expRNN [21]	98.7	96.6	-
UR-LSTM	99.28	96.96	71.00
UR-GRU [16]	99.27	96.51	74.4
LMU [42]	-	97.15	-
HiPPO-RNN [15]	98.9	98.3	61.1
UNIcoRNN [35]	-	98.4	-
LMUFFT [7]	-	98.49	-
LipschitzRNN [13]	99.4	96.3	64.2
S4	**99.63**	**98.70**	**91.13**

Figure 6: Image from the S4 appendix

S4 establishes SoTA on sCIFAR-10 with just 100,000 parameters (the authors don't specify the number for the other methods).

Let's conclude with a textual task and the benchmark *Long Range Arena* (LRA) by TAY et al. (2020).

Table 10: Full results for the Long Range Arena (LRA) benchmark for long-range dependencies in sequence models. (Top): Original Transformer variants in LRA. (Bottom): Other models reported in the literature.

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Random	10.00	50.00	50.00	10.00	50.00	50.00	36.67
Transformer	36.37	64.27	57.46	42.44	71.40	✗	53.66
Local Attention	15.82	52.98	53.39	41.46	66.63	✗	46.71
Sparse Trans.	17.07	63.58	59.59	44.24	71.71	✗	51.03
Longformer	35.63	62.85	56.89	42.22	69.71	✗	52.88
Linformer	35.70	53.94	52.27	38.56	76.34	✗	51.14
Reformer	37.27	56.10	53.40	38.07	68.50	✗	50.56
Sinkhorn Trans.	33.67	61.20	53.83	41.23	67.45	✗	51.23
Synthesizer	36.99	61.68	54.67	41.61	69.45	✗	52.40
BigBird	36.05	64.02	59.29	40.83	74.87	✗	54.17
Linear Trans.	16.13	65.90	53.09	42.34	75.30	✗	50.46
Performer	18.01	65.40	53.82	42.77	77.05	✗	51.18
FNet	35.33	65.11	59.61	38.67	77.80	✗	54.42
Nyströmformer	37.15	65.52	79.56	41.58	70.94	✗	57.46
Luna-256	37.25	64.57	79.29	47.38	77.72	✗	59.37
S4 (original)	58.35	76.02	87.09	87.26	86.05	88.10	80.48
S4 (updated)	**59.60**	**86.82**	**90.90**	**88.65**	**94.20**	**96.35**	**86.09**

Figure 7: Image from the S4 appendix

The LRA consisted of 6 tasks, including Path-X with a length of 16K tokens, for which the S4 was the first model to succeed, demonstrating its performance on very long-sequence tasks.
It would be more than 2 years before AMOS et al. showed in their paper *Never Train from Scratch: Fair Comparison of Long-Sequence Models Requires Data-Driven Priors* (2023) that transformers, introduced by Ashish VASWANI et al. (2017), (and not hybridized with an SSM) could also solve this task. However, unlike SSMs, they are unable to pass the 65K token PathX-256.

Note, a negative point concerning the text for S4: it obtains a higher perplexity compared to that of a transformer (standard, with more optimized versions having an even lower perplexity) on WikiText-103 by MERITY et al. (2016).

Table 8: (**WikiText-103 language modeling**) S4 approaches the performance of Transformers with much faster generation. (*Top*) Transformer baseline which our implementation is based on, with attention replaced by S4. (*Bottom*) Attention-free models (RNNs and CNNs).

Model	Params	Test ppl.	Tokens / sec
Transformer	247M	**20.51**	0.8K (1×)
GLU CNN	229M	37.2	-
AWD-QRNN	151M	33.0	-
LSTM + Hebb.	-	29.2	-
TrellisNet	180M	29.19	-
Dynamic Conv.	255M	25.0	-
TaLK Conv.	240M	23.3	-
S4	249M	**20.95**	**48K (60×)**

Figure 8: Image from the S4 appendix

This is probably due to the non-continuous nature of text (it has not been sampled from an underlying physical process such as speech or time series). We'll see in the article devoted to developments in SSM in 2023 that this point has been the subject of a great deal of work, and that SSM has now succeeded in bridging this gap.

Conclusion

SSMs are models with three views. A continuous view, and when discretized, a recurrent as well as a convolutive view.
The challenge with this type of architecture is to know when to favor one view over another, depending on the stage of the process (training or inference) and the type of data being processed.
This type of model is highly versatile, since it can be applied to text, vision, audio and time-series tasks (or even graphs).
One of its strengths is its ability to handle very long sequences, generally with a lower number of parameters than other models (ConvNet or *transformers*), while

still being very fast.
As we'll see in later article, the main differences between the various existing SSM architectures lie in the way the basic SSM equation is discretized, or in the definition of the $\mathbf{A}$ matrix.

To dig deeper

SSM history

Published two years earlier than S4, in December 2019, the LMU by VOELKER, KAJIĆ and ELIASMITH can be considered the ancestor of S4. In this paper, the authors initiate the recurrent view by proposing an alternative to HOCHREITER and SCHMIDHUBER's LSTM, which suffers from the problem of gradient vanishing when the number of processed steps becomes too high (limited to between 100 and 5000 depending on the variants). In the paper, they show that their model is capable of handling more than 100,000 steps (VOELKER even went up to over 1,000,000,000 steps in section 6.1 of his thesis). To do this, they base use the ODE $x'(t) = Ax(t) + Bu(t)$ (in the paper, x is denoted m), which they discretize via Euler method. The matrices A and B are obtained via Padé approximant, which strongly inspired the HiPPO framework. The key property of this dynamical system is that x represents sliding windows of u via Legendre polynomials up to degree d - 1. We invite the reader to consult section 2 of the paper for full details.
As indicated in the introduction, this paper is an application to deep learning of a more neuroscience-oriented model published in 2018 by the same authors.
Let's conclude by mentioning a sequel to the LMU work dating from February 2021 by CHILKURI and ELIASMITH. In this paper, they show how to compute their model efficiently. To do this, they parallelize the training by rewriting their ODE non-sequentially (see page 3 of the paper in particular), making it possible to use standard control-theoretic tools (see equation 22 of the paper and ÅSTRÖM and

MURRAY for full details) and then see things as well a convolution. They obtain better results than DistillBERT by SANH et al. (2019) with half as many parameters and doing character level modeling of the text8 dataset. Note also that the authors discretize their SSM via ZOH (Zero Order Hold), to which we'll return in more detail in the next blog post.

SSM ressources

To find out more about SSM, take a look at :

- The course (in French) on dynamic systems by Ion HAZYUK, Maitre de Conferences at INSA Toulouse (the part on state-space models starts from section 5.2)
- The doctoral thesis of Albert GU
- The doctoral thesis of Aaron R. VOELKER

S4 ressources

For S4, please consult the following resources:

- Videos:
  - Efficiently Modeling Long Sequences with Structured State Spaces - Albert Gu - Stanford MLSys #46 by Albert GU
  - MedAI #41: Efficiently Modeling Long Sequences with Structured State Spaces by Albert GU (a little longer, as more examples are covered)
  - JAX Talk: Generating Extremely Long Sequences with S4 by Sasha RUSH + the slides used in the video

- Codes:
  - The Annotated S4 (in Jax) by Sasha RUSH and Sidd KARAMCHETI
  - The GitHub of the official S4 implementation (in PyTorch)
  - Code of the Appendix E of the book "Hands-On Machine Learning with Scikit-Learn and Pytorch" by Aurélien Geron
- Blog posts:
  - Articles on S4 from the Hazy Research blog, which is the Stanford research group where Albert Gu did his PhD; part, part 2 and part 3.
- Book :
  - Appendix E of the book "Hands-On Machine Learning with Scikit-Learn and Pytorch" by Aurélien Geron

HiPPO ressources

For more information on the **HiPPO** matrix, please consult the following resources:

- The Hazy Research blog post on the subject
- The paper *How to Train Your HiPPO: State Space Models with Generalized Orthogonal Basis Projections* by Albert GU et al. (2022)

References

- Long short-term memory by Sepp HOCHREITER, Jürgen SCHMIDHUBER (1997)

- Feedback Systems by Karl Johan ÅSTRÖM, Richard M. MURRAY (2012 version)
- Learning Multiple Layers of Features from Tiny Images by Alex KRIZHESKY (2009)
- Pointer Sentinel Mixture Models by Stephen MERITY, Caiming XIONG, James BRADBURY, Richard SOCHER (2016)
- *Attention is all you need* by Ashish VASWANI, Noam SHAZEER, Niki PARMAR, Jakob USZKOREIT, Llion JONES, Aidan N. GOMEZ, Lukasz KAISER, Illia POLOSUKHIN (2017)
- Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition by Pete WARDEN (2018)
- Improving Spiking Dynamical Networks: Accurate Delays, Higher-Order Synapses, and Time Cells by Aaron R. VOELKER, Chris ELIASMITH (2018)
- Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks by Aaron R. VOELKER, Ivana KAJIĆ, Chris ELIASMITH (2019)
- Dynamical Systems in Spiking Neuromorphic Hardware by Aaron R. VOELKER (2019)
- DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter by Victor SANH, Lysandre DEBUT, Julien CHAUMOND, Thomas WOLF (2019)
- Long Range Arena: A Benchmark for Efficient Transformers by Yi TAY, Mostafa DEHGHANI, Samira ABNAR, Yikang SHEN, Dara BAHRI, Philip PHAM, Jinfeng RAO, Liu YANG, Sebastian RUDER, Donald METZLER (2020)
- Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting by Haoyi ZHOU, Shanghang ZHANG, Jieqi peng, Shuai ZHANG, Jianxin LI, Hui XIONG, Wancai ZHANG (2020)
- HiPPO: Recurrent Memory with Optimal Polynomial Projections by Albert GU, Tri DAO, Stefano ERMON, Atri RUDRA, Christopher RÉ (2020)
- Parallelizing Legendre Memory Unit Training by Narsimha CHILKURI, Chris ELIASMITH (2021)

- Combining Recurrent, Convolutional, and Continuous-time Models with Linear State-Space Layers by Albert GU, Isys JOHNSON, Karan GOEL, Khaled SAAB, Tri DAO, Atri RUDRA, Christopher RÉ (2021)
- Efficiently Modeling Long Sequences with Structured State Spaces by Albert GU, Karan GOEL, Christopher RÉ (2021)
- How to Train Your HiPPO: State Space Models with Generalized Orthogonal Basis Projections by Albert GU, Isys JOHNSON, Aman TIMALSINA, Atri RUDRA, Christopher RÉ (2022)
- On the Parameterization and Initialization of Diagonal State Space Models by Albert GU, Ankit GUPTA, Karan GOEL, Christopher RÉ (2022)
- Modeling sequences with structured state spaces by Albert GU (2023)
- Never Train from Scratch: Fair Comparison of Long-Sequence Models Requires Data-Driven Priors by Ido AMOS, Jonathan BERANT, Ankit GUPTA (2023)

👏 Community

Pom-Pom-Tom Sep 28

Holy crap just discovered this, what an absolutely excellent write-up and collection of resources. This feels like a complete one-stop shop for all things SSM. Amazing contribution - thank you!

1 reply · 🤗 1 +

lbourdois Article author Sep 30

Oh, that's unexpected. When I posted this article, comments weren't available under HF blog posts, so I wasn't expecting to receive any. Thank you for your kind words 😀

Reply in thread

ageron Oct 21

Thanks for this nice post. I'm confused by the HiPPO matrix equation in your post. Looking at the HiPPO paper, the HiPPO matrix for HiPPO-LegS is:

- sqrt(2n + 1) * sqrt(2k + 1) if k < n
- n + 1 if k = n
- 0 if k > n

I couldn't find a source for your version of the HiPPO matrix, which uses (–1)^(n–k) * (2k + 1) if k < n and k + 1 if k = n. Is it an error, or are these matrices somehow equivalent?

1 reply · 🤗 1 +

lbourdois Article author Oct 22

Thanks for the feedback!

Regarding the HiPPO matrix, in an earlier version of the article, I went into much more detail (also in relation to your following comment). Since I wanted to propose an introduction, I didn't want to include too much math and proof in this section, especially since this matrix is no longer used in practice in favor of diagonal matrices, which are much easier to calculate.

To come back to your question, the HiPPO-LegS formula you mentioned is correct. The one I list is actually HiPPO-LegT. And in practice, there is even HiPPO-FouT. Keeping track of all the different versions was quite complex at the time.
As I said, it's not very important to focus on this point, as we use much simpler things nowadays.

Nevertheless, if you're interested, I invite you to consult Figure 4 available in the section https://huggingface.co/blog/lbourdois/ssm-2022#s4-v2 of the blog article that followed this introduction. From a visual perspective, https://hazyresearch.stanford.edu/static/posts/2020-12-05-hippo/instantiations.png may also be helpful.

Over the weekend, I will take the time to clarify HiPPO-LegT rather than simply HiPPO and add a link to the following blog post.

👍 1 +

Reply in thread

ageron Oct 21

Just a nit-pick regarding the sentence: "*The authors prove in their paper that this type of matrix can be computed efficiently via three techniques (see Algorithm 1 in the paper): truncated generating series, Cauchy kernels and Woodbury identity.*"

I don't think that computing the HiPPO matrix itself is the problem, it's computing all of its powers A^i with i=0 to L (where L is the length of the input sequence). These powers are needed to build the kernel **K** in the convolutional representation. The authors actually avoided the problem by directly working in the complex frequency domain.

Also, I'm not sure what the following sentence means: "*Details of the demonstration showing that an NPLR matrix can be computed efficiently as a diagonal matrix can be found in the appendix (see part B and C) of the paper.*" Do you mean that the powers of an NPLR matrix can be computed efficiently, just like the powers of a diagonal matrix? I couldn't find which sections of part B and C you were referring to (was it in the LSSL paper?)

Lastly, you wrote "In the convolution kernel developed above, $\bar{C}$ and $\bar{B}$, are learnable scalars." However, I believe that $\bar{B}$ is a matrix of shape [n, 1] (i.e., a column vector) and $\bar{C}$ is a matrix of shape [1, n] (i.e., a row vector), where n in the state dimension.

Hope this helps.

1 reply ·

lbourdois Article author Oct 22 • edited Oct 22

Regarding the first comment, the “computed efficiently” may be misleading because it suggests that the calculations could be complex to perform if the method described in Algorithm 1 is not followed, when in fact it is not A itself that is complicated but the power of A as you understand.
One particularly important point is that this is a way of obtaining a HiPPO matrix with good coefficients. A HiPPO matrix is a normal + low-rank matrix where, under the hood, Legendre coefficients are tracked and everything can be calculated using FFT.
If one of the components is removed or if the matrix is initialized incorrectly (e.g., randomly), the *model will not converge*.

For the second point, I demonstrated in my v1 of the blog article that diagonal matrix = normal matrix + low rank. I hope to be able to find it over the weekend. Otherwise, I refer you to the LSSL paper (parts B and C). For a French person, there may be a few difficulties because we don't necessarily use the same theorem names as English speakers (I was confused by Picard-Lindelöf, which I had never heard of in my life, whereas in France I had actually seen it under the name Cauchy-Lipschitz theorem, also the names of discretizations but I think that's okay and also some notation from memory).

And you are absolutely right about the last point. I am no longer sure whether referring to a scalar is a misuse of language on my part, rather than referring to a vector (row or column matrix), or whether I had adopted the terminology used by the original authors so that if a reader reads my article and the publications in parallel, the notation would be the same.

I will take your comments into account to make this HiPPO part clearer over the weekend. Thanks 🤗

👍 1 +

Reply in thread

Edit Preview

Start discussing this article

Tap or paste here to upload images

Comment

Sign up or log in to comment