State Space Models for Time Series

Architecture, Mathematics, and Implementation

Leandro Stival

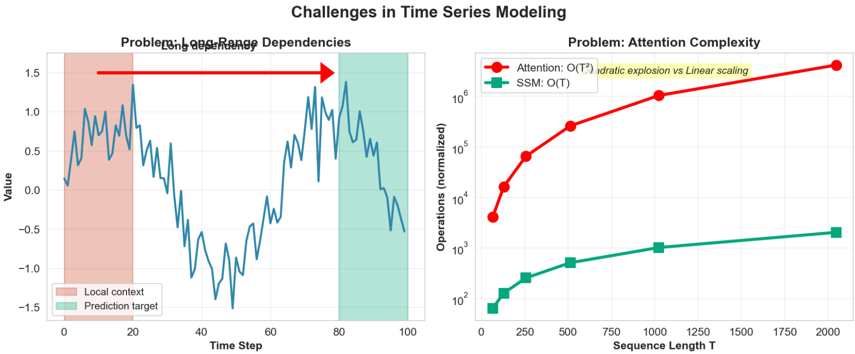
Wageningen University & Research

January 21, 2026

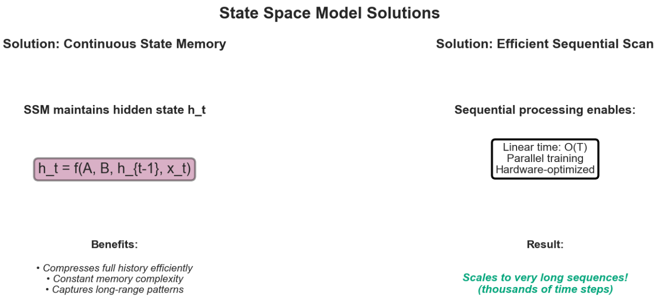
Overview

1. Motivation
2. Encoder Architecture
3. The Mamba Block
4. SSM Mathematics
5. Data Processing (Recurrence Plots)
6. Algorithm

Why State Space Models? (1/3)



Why State Space Models? (2/3)

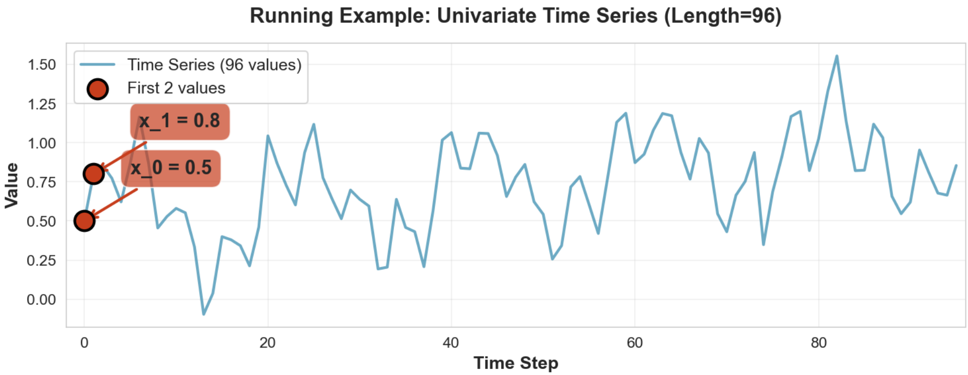


Why State Space Models? (3/3)

Key Advantages of SSMs:

- **Efficient Memory**: Compresses long history into fixed-size state
  - Maintains hidden state h_t that encodes full sequence history
  - Constant memory complexity regardless of sequence length
- **Linear Complexity**: $O(T)$ vs $O(T^2)$ for attention
  - Sequential processing enables efficient scanning
  - Scales to very long sequences (thousands of time steps)
- **Long Dependencies**: Designed for sequences with distant correlations
  - HiPPO initialization optimizes for long-range memory
  - Captures patterns across entire time series

Our Running Example (1/2)



Our Running Example (2/2)

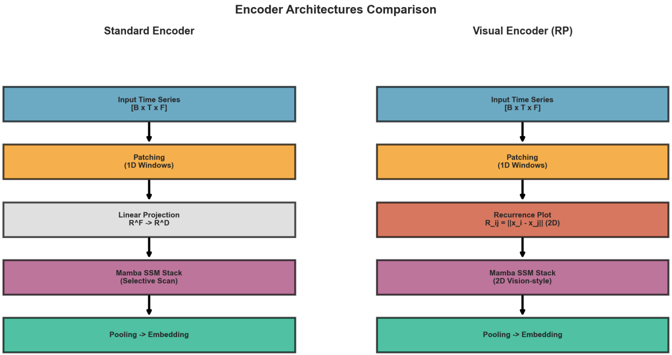
Throughout this presentation:

- We'll follow a **univariate time series** of length $T = 96$
  - Real-world time series with trend, seasonality, and noise
  - Representative of typical forecasting scenarios
- Focus on the **first 2 values**: $x_0 = 0.5$ and $x_1 = 0.8$
  - These specific values will appear in all numerical examples
  - Allows tracking data flow through the entire pipeline
- **Goal**: Understand how SSMs process sequential data
  - From raw input to final embedding
  - Step-by-step mathematical transformations

High-Level Encoder Architecture

- **Objective**: Map time series $X \in \mathbb{R}^{B \times T \times F}$ to compact embeddings $E \in \mathbb{R}^{B \times D_{emb}}$.
- **Core Components** (from `mamba_encoder.yaml`):
  1. **Tokenization**: Slicing time series into windows.
  2. **Projection**: Mapping tokens to model dimension D_{model}.
  3. **Backbone**: Stack of L Mamba Blocks (SSM).
  4. **Pooling**: Aggregating sequence info (Mean, Last, or CLS).

Encoder Variants (1/2)



Encoder Variants (2/2)

Standard Encoder

- **Input**: Raw sequence values
- **Patching**: Direct 1D windows
- **Projection**: Linear layer (`nn.Linear`)
- **Processing**: Sequential SSM blocks
- **Use Case**: General time series forecasting

Visual Encoder

- **Input**: Time series $\rightarrow$ pseudo-images
- **Transform**: Recurrence Plot
  - $R_{ij} = \|x_i - x_j\|$
- **Projection**: 2D convolution (`Conv2d`)
- **Processing**: Vision-style SSM
- **Use Case**: Capturing structural patterns

Inside the Mamba Block

The `MambaBlock` handles the sequence mixing.

Forward Pass (Simplified):

1. **Input Projection**: $x \in \mathbb{R}^{B \times T \times D} \rightarrow z, x' \in \mathbb{R}^{B \times T \times E}$
2. **Convolution**: 1D causal conv on x'
3. **SSM Processing**: `_selective_scan`(x', Δ, A, B, C)
4. **Output Projection**: Combine with gating z and project back

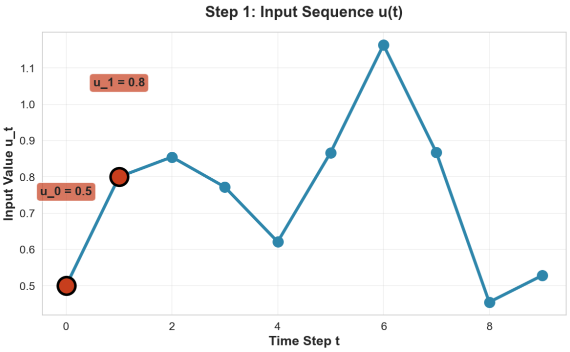
Selective Parameters: Δ, B, C are *input-dependent*. This is the core innovation!

Step 1: Input Sequence

The time series $\mathbf{x}$ enters the model. Each time step is a scalar (univariate) or vector (multivariate).

$$\mathbf{u} : \mathbb{R} \to \mathbb{R}^F$$

For our example: $u_0 = 0.5$, $u_1 = 0.8$, ...

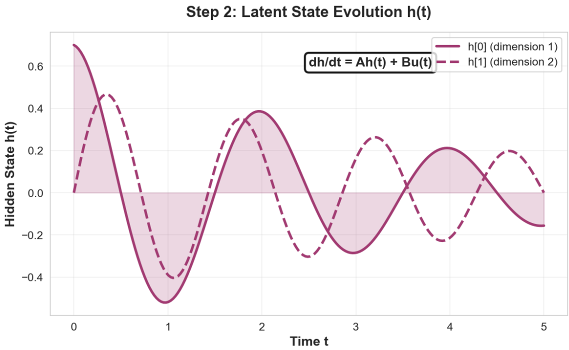


Step 2: Latent State $h(t)$

SSM maintains a **continuous hidden state** that captures history.

$$\frac{dh}{dt} = \mathbf{A}h(t) + \mathbf{B}u(t)$$

$\mathbf{A}$ encodes state dynamics (HiPPO-initialized).
$\mathbf{B}$ maps input influence.



Discretization (Zero-Order Hold)

To process sampled data with dynamic step sizes Δ, we discretize the continuous system. Given a step size Δ, the discrete parameters $\overline{\mathbf{A}}$ and $\overline{\mathbf{B}}$ are:

$$\overline{\mathbf{A}} = \exp(\Delta \mathbf{A}) \tag{1}$$

$$\begin{aligned} \overline{\mathbf{B}} &= (\Delta \mathbf{A})^{-1}(\exp(\Delta \mathbf{A}) - \mathbf{I}) \cdot \Delta \mathbf{B} \\ &= \mathbf{A}^{-1}(\overline{\mathbf{A}} - \mathbf{I}) \cdot \mathbf{B} \end{aligned} \tag{2}$$

This matches the specific implementation in `mamba_block.py`:

```
integral = torch.linalg.solve(A_expand, A_expm - eye)
B_disc = torch.bmm(integral, B_expand)
```

Section Summary: SSM Mathematics (1/2)

Continuous System

$\frac{dh}{dt} = \mathbf{A}h + \mathbf{B}u$ (Differential equation)

Models smooth evolution of hidden state over continuous time

Discretization (Zero-Order Hold)

$\overline{\mathbf{A}} = \exp(\Delta\mathbf{A}), \quad \overline{\mathbf{B}} = \mathbf{A}^{-1}(\overline{\mathbf{A}} - \mathbf{I})\mathbf{B}$

Converts continuous dynamics to discrete time steps of size Δ

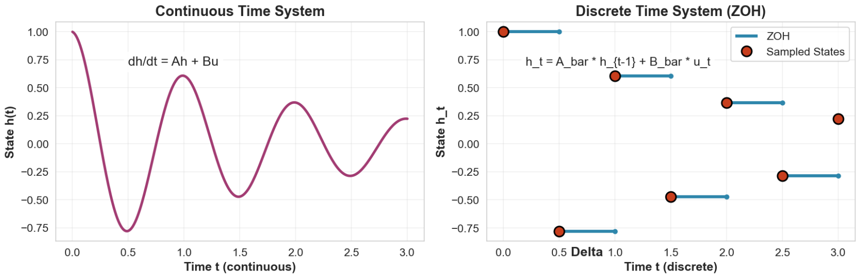
Discrete Recurrence

$h_t = \overline{\mathbf{A}}h_{t-1} + \overline{\mathbf{B}}u_t$ (State update)

$y_t = \mathbf{C}h_t$ (Output projection)

Enables efficient sequential processing of sampled data

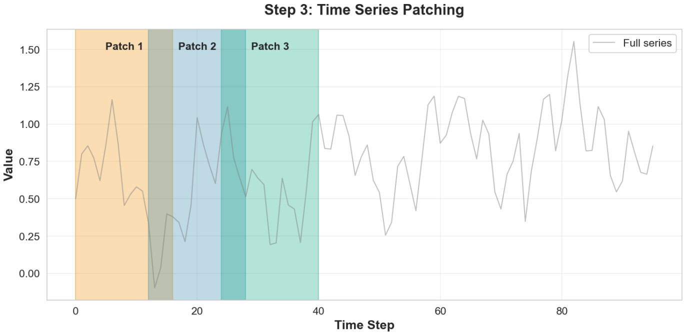
Section Summary: SSM Mathematics (2/2)



Key Concept: Zero-Order Hold (ZOH) maintains input constant between samples

Step 3: Time Series Patching

The raw time series is typically long and noisy.
We extract **overlapping patches** of fixed length (e.g., 96 steps).
This is analogous to image patching in Vision Transformers.

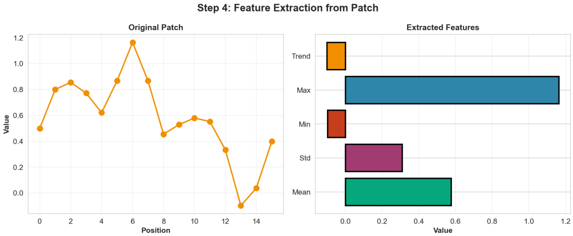


Step 4: Feature Extraction

From each patch, we can extract:

- **Raw values**: Direct encoding
- **Statistical features**: Mean, variance, etc.
- **Structural features**: Recurrence structure

The Visual Encoder uses the last option.



Step 5: Visual Transformation (RP)

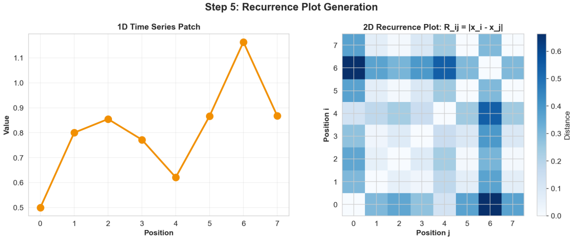
We convert the 1D patch into a 2D **Recurrence Plot**.

$$R_{i,j} = \|\mathbf{x}_i - \mathbf{x}_j\|$$

Numerical Example: Sequence patch (first 2 values): $\mathbf{x} = [0.5, 0.8]$

$$\mathbf{R} = \begin{pmatrix} |0.5 - 0.5| & |0.5 - 0.8| \\ |0.8 - 0.5| & |0.8 - 0.8| \end{pmatrix}$$

$$\mathbf{R} = \begin{pmatrix} 0 & 0.3 \\ 0.3 & 0 \end{pmatrix}$$



Algorithm: Selective Scan

Logic verified in: `src/models/mamba_block.py`

Algorithm 1 Selective Scan with HiPPO-Initialized SSM

Require: Input $\mathbf{x} \in \mathbb{R}^{B \times T \times D}$, step sizes $\boldsymbol{\delta} \in \mathbb{R}^{B \times T \times 1}$
Ensure: Output $\mathbf{y} \in \mathbb{R}^{B \times T \times D}$

Precompute $\mathbf{A}, \mathbf{B}, \mathbf{C}$ (HiPPO)
for $t = 0$ **to** $T - 1$ **do**
  $\mathbf{u}_t \leftarrow \mathbf{x}[:, t, :]$; $\Delta_t \leftarrow \boldsymbol{\delta}[:, t, :]$
  $\widetilde{\mathbf{A}} \leftarrow \Delta_t \cdot \mathbf{A}$ {Scale A by time step}
  $\mathbf{A}_d \leftarrow \exp(\widetilde{\mathbf{A}})$ {Discretize A}
  $\mathbf{h} \leftarrow \mathbf{A}_d\,\mathbf{h} + \mathbf{B}_d\,\mathbf{u}_t$ {Update State}
  $\mathbf{y}_t \leftarrow \mathbf{C}\,\mathbf{h}$ {Project to Output}
end for

Numerical Walkthrough: Selective Scan (1/2)

Numerical Walkthrough: Selective Scan (2/2)

Key Insight: At $t = 1$, state h_1 retains memory from both u_0 and u_1!

- **Input Sequence**:
  - $u_0 = 0.5$, $u_1 = 0.8$ (from our running example)
  - Step size: $\Delta = 0.5$
- **State Evolution**:
  - $h_0 = \overline{\mathbf{B}}u_0 \approx [0.195, 0.145]^T$
  - $h_1 = \overline{\mathbf{A}}h_0 + \overline{\mathbf{B}}u_1 \approx [0.431, 0.318]^T$
  - State accumulates information: h_1 depends on both u_0 and u_1
- **Output Sequence**:
  - $y_0 = \mathbf{C}h_0 = 0.195$
  - $y_1 = \mathbf{C}h_1 = 0.431$

Implementation Specifics

Based on `mamba_block.py`:

- **HiPPO Initialization**: Matrix $\mathbf{A}$ is initialized using the Legendre-S (LegS) measure to handle long-term dependencies.
- **Parallelism vs. Scanning**:
  - The Python implementation performs a **sequential loop** (Line 144 in `mamba_block.py`).
  - Optimized CUDA implementations (not currently used) usually perform a parallel associative scan.
- **Numerical Stability**:
  - Δ is clamped: $\Delta \in [10^{-4}, 3.0]$.
  - `torch.linalg.solve` used instead of explicit inverse for $\mathbf{A}^{-1}$ to ensure stability.

Summary: State Space Models for Time Series

Core Concepts:

1. **Motivation**: Efficient long-range dependencies
2. **SSM Math**: Continuous $\rightarrow$ Discrete via ZOH
3. **Selective Scan**: Dynamic state updates
4. **Visual Encoding**: Recurrence Plots for 2D patterns

Implementation:

- HiPPO initialization for $\mathbf{A}$
- Parallel training + Sequential inference
- Two encoder variants: Standard & Visual
- Numerically stable via `torch.linalg.solve`

Running Example: Time series $(T = 96)$ with $x_0 = 0.5, x_1 = 0.8$
$\Rightarrow$ State memory: h_1 encodes both past and present!

Questions?