Lecture 16, 17: State Space Models (SSM) - From S4 to Mamba

Contents

1 Introduction and Motivation

Traditional sequence modeling has historically faced a dichotomy between two dominant architectures: Recurrent Neural Networks (RNNs) and Transformers (Attention).

- **RNNs:** They possess a concept of "state" (h_t), allowing them to theoretically capture infinite history with constant inference time ($O(1)$ per step). However, they suffer from the "horizontal nonlinearity" problem. The sequential dependence on the previous hidden state through a nonlinearity prevents parallelization during training, leading to slow training on long sequences. Furthermore, they struggle with the vanishing gradient problem, making it difficult to learn long-range dependencies.
- **Transformers:** They utilize attention mechanisms to model dependencies directly. While highly parallelizable during training, they suffer from quadratic complexity $O(T^2)$ with respect to sequence length T and have a linear inference cost $O(T)$ per step (KV cache), making them inefficient for very long contexts.

State Space Models (SSMs) emerge as a solution that aims to combine the best of both worlds:

1. **Training:** $O(T \log T)$ or $O(T)$ parallelizable training (like CNNs/Transformers).
2. **Inference:** $O(1)$ constant time inference (like RNNs).
3. **Performance:** Ability to capture very long-range dependencies.

2 The Linear State Space Model (S4)

The core innovation of modern SSMs (like S4) is to remove the nonlinearity in the state transition, relying on a **Linear Time Invariant (LTI)** system.

2.1 Continuous-Time Formulation

The model is inspired by a continuous-time latent state model mapping a 1-D input signal $x(t)$ to a 1-D output signal $y(t)$ through a latent state $h(t) \in \mathbb{R}^N$:

$$\dot{h}(t) = \mathbf{A}h(t) + \mathbf{B}x(t) \tag{1}$$

$$y(t) = \mathbf{C}h(t) + \mathbf{D}x(t) \tag{2}$$

Where:

- $\mathbf{A} \in \mathbb{R}^{N \times N}$ is the evolution matrix.
- $\mathbf{B} \in \mathbb{R}^{N \times 1}$ is the input matrix.
- $\mathbf{C} \in \mathbb{R}^{1 \times N}$ is the output matrix.
- $\mathbf{D} \in \mathbb{R}^{1 \times 1}$ is the skip connection.

2.2 Discrete-Time Recurrence (RNN View)

To operate on discrete sequence data, the continuous system must be discretized (typically using Zero-Order Hold, detailed in Section 4.3). This yields the recurrence:

$$h_t = \overline{\mathbf{A}}h_{t-1} + \overline{\mathbf{B}}x_t \tag{3}$$

$$y_t = \mathbf{C}h_t + \mathbf{D}x_t \tag{4}$$

This looks like a linear RNN. The linearity is crucial because it allows us to unroll the recurrence efficiently.

2.3 The Convolutional View

Consider the expansion of the hidden state for a sequence starting at $h_{-1} = 0$:

$$\begin{aligned} h_0 &= \overline{\mathbf{B}}x_0 \\ h_1 &= \overline{\mathbf{A}}\overline{\mathbf{B}}x_0 + \overline{\mathbf{B}}x_1 \\ h_2 &= \overline{\mathbf{A}}^2\overline{\mathbf{B}}x_0 + \overline{\mathbf{A}}\overline{\mathbf{B}}x_1 + \overline{\mathbf{B}}x_2 \end{aligned}$$

The output y_k (ignoring $\mathbf{D}$ for brevity) is:

$$y_k = \sum_{j=0}^{k} \mathbf{C}\overline{\mathbf{A}}^{k-j}\overline{\mathbf{B}}x_j \tag{5}$$

This operation is essentially a discrete convolution:

$$y = \mathbf{K} * x \tag{6}$$

Where $\mathbf{K}$ is the **SSM Kernel**:

$$\mathbf{K} = (\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{A}}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{A}}^2\overline{\mathbf{B}}, \ldots, \mathbf{C}\overline{\mathbf{A}}^{T-1}\overline{\mathbf{B}}) \tag{7}$$

Key Insight 1. *Because the system is linear, we can compute the entire output sequence y from input x in parallel using convolution, avoiding the sequential bottleneck of RNN training.*

3 Efficient Computation and Stability

While the convolutional view allows parallelization, a naive convolution is $O(T^2)$. To make this efficient for deep learning, several modifications are required.

3.1 Fast Fourier Transform (FFT)

By the Convolution Theorem, convolution in the time domain is multiplication in the frequency domain.

$$y = \text{IFFT}(\text{FFT}(\mathbf{K}) \cdot \text{FFT}(x)) \tag{8}$$

This reduces the computational complexity from $O(T^2)$ to $O(T \log T)$.

3.2 Handling Vector Inputs (Depthwise Convolution)

In deep learning, inputs x_t are vectors of dimension D, not scalars.

- A full mixing matrix would imply $D \times D$ kernels, leading to massive computation (10,000 convolutions for $D = 100$).
- **Solution:** Analogous to depthwise separable convolutions in CNNs, SSMs usually operate **independently per channel**.
- Each dimension of the input has its own independent dynamics $(\mathbf{A}, \mathbf{B}, \mathbf{C})$. Mixing across channels occurs in subsequent projection layers (like MLPs), not within the SSM block itself.

3.3 Structured Matrix A (Diagonalization)

Computing the kernel $\mathbf{K}$ involves powers of $\overline{\mathbf{A}}$. This is expensive for general matrices.

- **Constraint:** We restrict $\mathbf{A}$ to be a **Diagonal Matrix**.
- Diagonal matrices are easy to exponentiate. This restriction does not significantly reduce expressivity because most matrices are diagonalizable (except those with non-trivial Jordan blocks), and we are learning the parameters.

3.4 Long-Range Dependencies and Initialization

For the model to retain history, the eigenvalues of $\overline{\mathbf{A}}$ (denoted λ) are critical.

- $|\lambda| > 1$: System explodes (unstable).
- $|\lambda| < 1$: System forgets history exponentially fast (vanishing gradient).
- $|\lambda| = 1$: Critical damping (preserves magnitude).

To capture long-range dependencies, eigenvalues should be close to the unit circle. **Hippo Initialization:** S4 uses specific matrix structures (Hippo) or initialization schemes where eigenvalues are complex numbers initialized near the unit circle.

$$\lambda = \exp(-\text{ReLU}(\lambda_{real}) + i\lambda_{imag}) \tag{9}$$

The ReLU ensures the real part is negative (in continuous time), ensuring stability (decaying rather than exploding), while the complex part allows for oscillatory behavior that retains memory.

4 Selectivity and Mamba (S6)

The LTI formulation of S4 has a major limitation: **It is content-independent.** The dynamics matrices $(\overline{\mathbf{A}}, \overline{\mathbf{B}})$ are fixed for all time steps. The model processes every token with the same "filter," regardless of context.

4.1 The Need for Selectivity

In tasks like "associative recall" or language modeling, the model needs to:

- Focus on relevant information (Selective Copying).
- Ignore irrelevant noise (Selective Ignoring).

An LTI system cannot adaptively reset its state or change its focus based on the current input x_t.

4.2 Input-Dependent Dynamics

The Mamba (S6) model introduces **Selectivity** by making the parameters functions of the input:

$$\mathbf{B} \to \mathbf{B}_t(x_t) \tag{10}$$
$$\mathbf{C} \to \mathbf{C}_t(x_t) \tag{11}$$
$$\Delta \to \Delta_t(x_t) \tag{12}$$

This transforms the model from LTI to **Linear Time-Varying (LTV)**. Consequently, the convolution theorem (FFT) no longer applies, as the kernel is not stationary. However, efficient parallel computation is still possible using **Parallel Scans** (prefix sums).

4.3 Discretization as a Gating Mechanism

To understand how selectivity works, we look at the discretization of the continuous system using the Zero-Order Hold (ZOH) method.

Given the continuous system $\dot{h}(t) = \mathbf{A}h(t) + \mathbf{B}x(t)$ and a sampling step Δ:

$$h(t+\Delta) = e^{\mathbf{A}\Delta}h(t) + \int_t^{t+\Delta} e^{\mathbf{A}(t+\Delta-\tau)}\mathbf{B}x(\tau)d\tau \tag{13}$$

Assuming inputs are constant during the interval Δ (Zero-Order Hold), the discrete parameters become:

$$\overline{\mathbf{A}} = \exp(\mathbf{A}\Delta) \tag{14}$$

$$\overline{\mathbf{B}} = (\mathbf{A}\Delta)^{-1}(\exp(\mathbf{A}\Delta) - \mathbf{I}) \cdot \Delta\mathbf{B} \approx \Delta\mathbf{B} \quad \text{(first order approx)} \tag{15}$$

4.3.1 The Role of Δ (Delta)

The parameter Δ acts as a gate based on the time-scale of the input:

- **Small Δ ($\to 0$):**
  - $\overline{\mathbf{A}} \to \mathbf{I}$ (Identity). The state persists.
  - $\overline{\mathbf{B}} \to 0$. The input is ignored.
  - **Effect:** Preserve history, ignore current input.
- **Large Δ:**
  - $\overline{\mathbf{A}} \to 0$ (assuming stable system with negative real eigenvalues). The state is reset.
  - $\overline{\mathbf{B}}$ becomes large.
  - **Effect:** Forget history, focus heavily on current input.

4.4 Implementing Selectivity

In Mamba, Δ_t is learned directly from the input x_t via a projection:

$$\Delta_t = \text{Softplus}(\text{Linear}(x_t)) \tag{16}$$

The Softplus function $\ln(1 + e^x)$ ensures Δ_t is positive. By learning to modulate Δ_t based on content, the model learns when to write to memory and when to reset it, effectively implementing a gating mechanism similar to LSTM gates but derived from first principles of continuous systems.

5 Summary of Architecture Evolution

1. **RNN:** $h_t = \sigma(Wh_{t-1} + Ux_t)$. Slow training, nonlinearity prevents parallelization.
2. **S4 (LTI SSM):** $h_t = \overline{\mathbf{A}}h_{t-1} + \overline{\mathbf{B}}x_t$.
   - Removes nonlinearity for parallel training via Convolution/FFT.
   - Uses diagonal $\mathbf{A}$ and complex initialization for long range.
   - *Limitation:* Cannot selectively attend to content (LTI).
3. **Mamba (Selective SSM):** $h_t = \overline{\mathbf{A}}(\Delta_t)h_{t-1} + \overline{\mathbf{B}}(\Delta_t)x_t$.
   - Makes Δ, B, C input-dependent.

- Sacrifices Convolution/FFT training.
- Uses **Parallel Scan** for efficient $O(T)$ training.
- Achieves "Transformer-quality" performance on language modeling tasks by solving the selectivity problem.