On the Linearization of Recurrent Modeling

Radu-Florin Ciobanu
radu-florin.ciobanu@s.unibuc.com

Abstract

This paper presents the reconsideration of sequential models in the current literature as a way to overcome the inherent problems and space complexity of Transformer-based models. Specifically, we discuss how well-optimized linear architectures can offer compelling advantages for resource-efficient sequence modeling, particularly for processing long sequences in constrained environments.

1 Introduction

In recent years, Transformers (Vaswani et al., 2017) have emerged as the leading architecture across various domains, driving significant progress in NLP tasks (Devlin et al., 2019). Despite their success, Transformers suffer from quadratic computational complexity with respect to sequence length, making them computationally expensive for long sequences, particularly in resource-constrained environments. As a result, many researchers have focused on developing more efficient models that can offer performance comparable to Transformers.

State space modeling has reemerged as a fundamental method for efficiently handling sequences in a more computationally efficient manner while preserving model expressiveness. By exploring these alternative approaches, we seek to emphasize the trade-offs between sequential processing and the parallelism provided by Transformers.

2 State Space Models

A *State Space Model* broadly refers to any recurrent process with a latent hidden state. It is applied across various disciplines to describe different concepts, starting from RNNs and ConvNets, to Markov Decision Processes (MDPs) and Kalman Filters (Kalman, 1960).

The classic recurrent architectures LSTM (Hochreiter, 1997) and GRU (Chung et al., 2014) had driven significant progress in many sequence related tasks, NLP (Sutskever, 2014) or speech processing (Graves et al., 2013). and they also laid the groundwork for the initial implementation of attention mechanisms (Bahdanau, 2014).

2.1 Structured State Space Models (S4)

Structured SSMs (S4) (Gu et al., 2022) are sequence-to-sequence mapping systems $x(t) \in \mathbb{R} \mapsto y(t) \in \mathbb{R}$ through an implicit hidden state $h(t) \in \mathbb{R}^d$. S4 consists of parameters $(\Delta \in \mathbb{R}^d, \mathbf{A} \in \mathbb{R}^{d \times d_{state}}, \mathbf{B} \in \mathbb{R}^{d \times d_{state}}, \mathbf{C} \in \mathbb{R}^{d \times d_{state}})$, where d is the embedding dimension and d_{state} determines the latent state dimension. The forward pass begins with discretization of parameters $(\mathbf{A}, \mathbf{B}) \mapsto (\bar{\mathbf{A}}, \bar{\mathbf{B}})$ by step Δ via zero-order hold (ZOH) discretization:

$$\Delta \leftarrow \tau_\Delta(\Delta) \tag{1}$$
$$\bar{\mathbf{A}} \leftarrow \exp(\Delta \mathbf{A}) \tag{2}$$
$$\bar{\mathbf{B}} \leftarrow (\Delta \mathbf{A})^{-1}(\exp(\Delta \mathbf{A}) - I) \cdot \Delta \mathbf{B} \tag{3}$$

where τ_Δ is softplus (Bridle, 1990) or any smooth ReLU. The SSM computes the hidden state and the output recursively as follows:

$$h_t \leftarrow \bar{\mathbf{A}} h_{t-1} + \bar{\mathbf{B}} x_t \tag{4}$$
$$y_t \leftarrow \mathbf{C} h_t \tag{5}$$

We observe that an SSM is similar with an RNN except it replaces the nonlinearity with another linear transformation, that can be computed recursively in $O(n)$, but the scalability strengths stays in the parallelization of this operation due the strict input dependence, by rewriting it as a global convolution that leverages the Fast Fourier Transform (FFT) as primitive, running in parallel in $O(n \log n)$ space complexity. To understand how, we show that any y_k can be computed as a dot product between a pre-computed vector consisting of

the actual constant through-time parameters and the input $x_1, x_2, \ldots, x_k$:

$$\begin{aligned} y_k &= \mathbf{C}(\bar{\mathbf{A}}h_k + \bar{\mathbf{B}}x_k) \\ &= \mathbf{C}(\bar{\mathbf{A}}(\bar{\mathbf{A}}h_{k-1} + \bar{\mathbf{B}}x_{k-1}) + \bar{\mathbf{B}}x_k) \\ &= \mathbf{C}(\bar{\mathbf{A}}(\bar{\mathbf{A}}(\bar{\mathbf{A}}h_{k-2} + \bar{\mathbf{B}}x_{k-2}) + \bar{\mathbf{B}}x_{k-1}) + \bar{\mathbf{B}}x_k) \\ &= \ldots \end{aligned}$$

This can be written as a dot product:

$$y_k = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{k-1} \\ x_k \end{pmatrix} \cdot \begin{pmatrix} \mathbf{C}\overline{\mathbf{A}\mathbf{A}\ldots\mathbf{A}}^{k-1}\overline{\mathbf{B}} \\ \mathbf{C}\overline{\mathbf{A}\mathbf{A}\ldots\mathbf{A}}^{k-2}\overline{\mathbf{B}} \\ \vdots \\ \mathbf{C}\overline{\mathbf{A}}\overline{\mathbf{B}} \\ \mathbf{C}\overline{\mathbf{B}} \end{pmatrix} \quad (6)$$

The entire output sequence $y = y_1, y_2, \ldots, y_k$ can be computed as a 1D convolution:

$$y \leftarrow x * \overline{\mathbf{K}} \text{ where } \overline{\mathbf{K}} \text{ is precomputed} \quad (7)$$

where $\overline{\mathbf{K}}$ is the second vector in Eq. 6.

Algorithm 1 S4. For simplicity $\Delta_{rank} = d$, but in actual implementations $\Delta_{rank} = d/c, c \in \mathbb{N}^*$.

Input: $x \in \mathbb{R}^{B \times L \times d}$
Output: $y \in \mathbb{R}^{B \times L \times d}$
Require: $\Delta \in \mathbb{R}^{\Delta_{rank}}, (\mathbf{A}, \mathbf{B}, \mathbf{C}) \in \mathbb{R}^{d \times d_{state}}$

1: $\bar{\mathbf{A}}, \bar{\mathbf{B}} \in \mathbb{R}^{B \times L \times \Delta_{rank} \times d_{state}} \leftarrow \text{ZOH}(\Delta', \mathbf{A}, \mathbf{B})$
2: $y \leftarrow SSM(\bar{\mathbf{A}}, \bar{\mathbf{B}}, \mathbf{C}, x)$ ▷ recurrence or conv
3: **return** y

2.2 Selective State Space Models (S6)

The S4 has *linear time invariance*, i.e. the parameters remain fixed throughout the timesteps (as in RNNs and ConvNets). Comparing to its update introduced in the Mamba architecture (Gu and Dao, 2023), despite its success in solving the Copying task (that involves constant spacing between the input and the output), it fails in the Selective Copying task, which has random spacing in between inputs. In order to solve it, they introduced time-varying parameters, i.e. a dynamic gating mechanism that remembers and ignores features of the embeddings of the input sequence. The selection works by converting the parameters into *time-varying parameters* - generate a different set of pairs $(\overline{\mathbf{A}}, \overline{\mathbf{B}}, \mathbf{C})_t$ for each timestep t that is input dependent.

To understand the similarity with GRU, let us consider an SSM in which $d_{inner} = 1$, $\mathbf{A} = -1$, $\mathbf{B} = 1$, and $\tau_\Delta = \log(1 + \exp x)$ as nonlinearity. In this case, following Eq. 4, the continuous (no discretized parameters) SSM will be to:

$$h_t \leftarrow -h_t + x_t \quad (8)$$

The zero-order hold discretization will be computed as:

$$\begin{aligned} \Delta_t &= \tau_\Delta(\Delta + x_t\Delta) \\ \overline{\mathbf{A}}_t &= \exp(\Delta_t\mathbf{A}) \\ &= \frac{1}{1 + \exp(\text{Linear}_\Delta(x_t))} \\ &= \sigma(-\text{Linear}_\Delta(x_t)) \\ &= 1 - \sigma(\text{Linear}_\Delta(x_t)) \\ \overline{\mathbf{B}}_t &= (\Delta_t\mathbf{A})^{-1}(\exp(\Delta_t A) - I) \cdot \Delta_t\mathbf{B} \\ &= -(\exp(\Delta_t\mathbf{A}) - I) \\ &= 1 - \overline{\mathbf{A}} \\ &= \sigma(\text{Linear}_\Delta(x_t)) \end{aligned}$$

Now it is clear that $\overline{\mathbf{A}}$ and $\overline{\mathbf{B}}$ represent the update gate in Eq. 4, which is similar to the GRU hidden state formula $h_t = (1 - z_t)h_{t-1} + z_t x_t$, where the gate z_t in this case is derived only from the input.

Algorithm 2 S6. In the Mamba block, the input x with embedding dimension d is expanded by ϵ, so $d_{inner} = d\epsilon$. The authors implementation is simplified comparing to the algorithm definition in the paper in terms of parameters dimension: default $\epsilon = 2$, the latent space dim $d_{state} = 16$, and the dimension of Δ is $\Delta_{rank} = d/16$.

Input: $x \in \mathbb{R}^{B \times L \times d_{inner}}$
Output: $y \in \mathbb{R}^{B \times L \times d_{inner}}$
Require: $\Delta \in \mathbb{R}^{\Delta_{rank}}$
Require: $\mathbf{A} \in \mathbb{R}^{d_{inner} \times d_{state}}$
Require: $\mathbf{B} \in \mathbb{R}^{d \times d_{state}}$
Require: $\mathbf{C} \in \mathbb{R}^{d \times d_{state}}$

1: $\mathbf{B}' \in \mathbb{R}^{B \times L \times d_{state}} \leftarrow x\mathbf{B}$
2: $\mathbf{C}' \in \mathbb{R}^{B \times L \times d_{state}} \leftarrow x\mathbf{C}$
3: $\Delta' \in \mathbb{R}^{B \times L \times \Delta_{rank}} \leftarrow \tau_\Delta(\Delta_{broadcast} + x\Delta)$
4: $\bar{\mathbf{A}}, \bar{\mathbf{B}} \in \mathbb{R}^{B \times L \times \Delta_{rank} \times d_{state}} \leftarrow \text{ZOH}(\Delta', \mathbf{A}, \mathbf{B})$
5: $y \leftarrow SSM(\bar{\mathbf{A}}, \bar{\mathbf{B}}, \mathbf{C}, x)$ ▷ reccurence (scan)
6: **return** y

Computing the final output y is no longer possible with global convolution because of the time-

varying parameters (expanded parameters for each element in the sequence), but it can still be computed efficiently in parallel with associative scan (see Section 3). A Mamba Block projects the input to a higher dimensionality by an expansion factor $\epsilon = 2$ before passing it through the S6, and reduces it back from $d\epsilon$ to d after a gate is induced:

Algorithm 3 Mamba (1) Block

Input: $x \in \mathbb{R}^{B \times L \times d}$
Output: $y \in \mathbb{R}^{B \times L \times d}$
Require: $\text{Linear}_{up}, \text{Linear}_{up}^{gate}, \text{Linear}_{down} \in \mathcal{R}^{d_{inner} \times d}$
Require: Conv1D & $S6$
Require: $\epsilon = 2$, expansion factor ($d_{inner} = d \cdot \epsilon$)

$x \in \mathbb{R}^{B \times L \times d_{inner}} \leftarrow \text{Linear}_{up}(x)$
$x_{gate} \in \mathbb{R}^{B \times L \times d_{inner}} \leftarrow \text{Linear}_{up}^{gate}(x)$
$x \leftarrow \text{Swish}(\text{Conv1D}(x))$
$x \leftarrow S6(x)$
$x \leftarrow x \odot \text{Swish}(x_{gate})$
$y \leftarrow \text{Linear}_{down}(x)$ ▷ $d_{inner} \rightarrow d$
return y

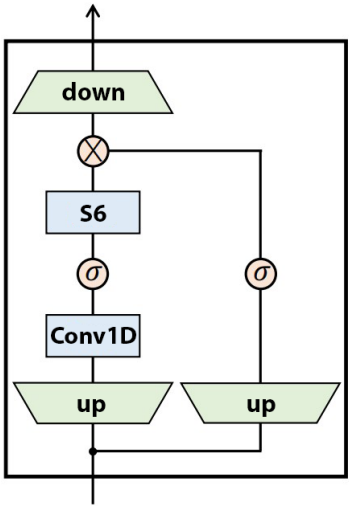


Figure 1: Mamba (1) Block (Gu and Dao, 2023), the sequence transformation is gated before down linear projection

The linear layers project the input x into a higher (multiple) dimension d_{inner}. In the authors implementation[1], the Convolution layer has $\text{channels}_{\text{in}} = \text{channels}_{\text{out}} = d_{inner}$, $\text{filter}_{\text{dim}} = 4$, no. groups $= d_{inner}$, and padding $= \text{filter}_{\text{dim}} - 1$ to retain the dimensionality. The model is constructed by stacking up multiple Mamba Blocks wrapped within residual blocks with pre-normalization.

Algorithm 4 Residual Block

Input: $x \in \mathbb{R}^{B \times L \times d}$
Output: $y \in \mathbb{R}^{B \times L \times d}$
Require: MambaBlock

$x_{residual} \leftarrow \text{LayerNorm}(x)$ ▷ or RMSNorm
$x_{residual} \leftarrow \text{MambaBlock}(x)$
$y \leftarrow x_{residual} + x$
return y

Further research (Dao and Gu, 2024) for more efficient large-scale training and inference was done, with a focus on Tensor Parallelism (TP) (Shoeybi et al., 2019) and Sequence Parallelism (SP) (Korthikanti et al., 2023). At first, Mamba (2) has multiple parallel groups g that process the input separately, similar to Multihead Attention (MHA) in Transformers, and the outputs of all heads pass through an all-reduce layer ($y \leftarrow \sum_g y_g$). ($\text{Linear}_{up}, \text{Linear}_{up}^{gate}, \text{Linear}_{down} \in \mathcal{R}^{d_{inner}/g \times d}$). The block design itself received some modifications - SSM parameters $(\mathbf{A}, \mathbf{B}, \mathbf{C})$ are produced at the beginning of the block instead of as a function of the SSM input and an additional GroupNorm (GN) (Wu and He, 2018) layer.

When working with a larger number of GPUs, the input can be split along the sequence (L) dimension. For example GPU-0 that computes the first L/gpu_{num} outputs y, can pass the hidden states to GPU-1. Using the hidden states of the previous tokens in the sequence and the next L/gpu_{num} tokens as inputs, GPU-1 computes the next output y of the same length, and its hidden states are passed further.

3 Parallel associative scan

A parallel log space implementation for the CumSum (Prefix sum) function was firstly introduced in (Blelloch, 1990) that lie at the foundation of linear RNNs efficiency (Martin and Cundy, 2018), more precisely, the algorithm can be generally applied for a family of functions of form $v_t = a_t \oplus v_{t-1} + b_t$, where $\oplus$ is any associative operator, $a_1, a_2, \ldots, a_3$ and $b_1, b_2, \ldots, b_3$ are the inputs, and $v_1, v_2, \ldots, v_3$ the outputs.

The naive implementation of the cumulative sum operation $s_t = v_{t-1} + v_t$ takes n steps of computation, though using multiple workers will drop down to a parallel log space complexity. The idea is to

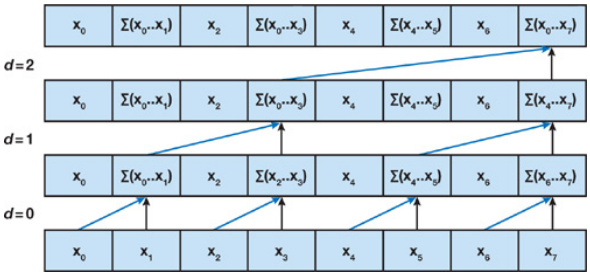
[1] https://github.com/state-spaces/mamba/blob/main/mamba$_s sm/modules/mamba2.py$

compute in a form of balanced binary tree with n leaves and a depth of $\log n$, where n is a power of 2 (for arbitrary sizes the vector is padded). The algorithm is split into two phases. The $up-sweep$ phase computes partial sums on even indexes, and on the indexes that are powers of two the complete for it's index:

Algorithm 5 Up-sweep

Input: $v \in \mathbb{R}^n$

for $d = 0 \rightarrow \log_2(n) - 1$ **do**
  for $w_0, w_1, \ldots, w_{\frac{n-1}{2^{d+1}}}$ **in parallel do**
    $i \leftarrow w_{index} \cdot 2^{d+1}$
    $v_{i+2^{d+1}-1} \leftarrow v_{i+2^d-1} \oplus v_{i+2^{d+1}-1}$
  end for
end for

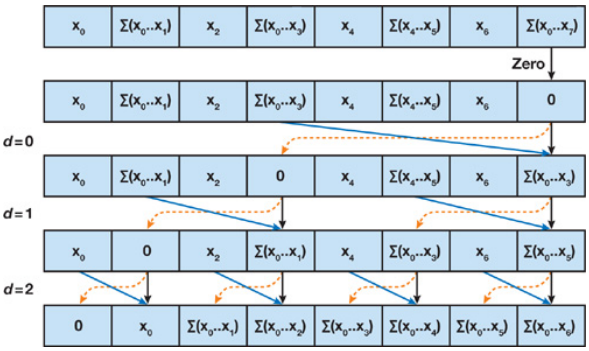


The $down-sweep$ phase sets the last element (the root of the tree) to 0 and traverses the tree from root to leaves to build the scan in the vector using the partials sums from the previous phase. In each step d, each node passes its own value to its left child, and the sum of its value and the former value of its left child to its right child.

Algorithm 6 Down-sweep

Input: $v \in \mathbb{R}^n$ ($up-phase$ result)
Output: $v \in \mathbb{R}^n$ (exclusive scan)

$v_{n-1} \leftarrow 0$
for $d = \log_2(n) - 1 \rightarrow 0$ **do**
  for $w_0, w_1, \ldots, w_{\frac{n-1}{2^{d+1}}}$ **in parallel do**
    $i \leftarrow w_{index} \cdot 2^{d+1}$
    $t \leftarrow v_{i+2^d-1}$
    $v_{i+2^d-1} \leftarrow v_{i+2^{d+1}-1}$
    $v_{i+2^{d+1}-1} \leftarrow t \oplus v_{i+2^{d+1}-1}$
  end for
end for
return v



Note that the result of the down-sweep is an exclusive scan, i.e. the vector is shifted to the right. Shifting it to the left and inserting at the end the sum of the last element of the exclusive scan and the last element in the initial array results in an inclusive scan. The presentation of the base prefix sum algorithm, including the illustrations were taken from an official NVIDIA documentation[2] explained after (Blelloch, 1990), where they explain hardware aware implementations. Now let's apply the algorithm (cumsum and cumprod) to compute the hidden states for the functions $h_t = a_t \cdot h_{t-1} + b_t, h_0 \leftarrow b_0$. Following (Heinsen, 2023), we leverage the complex plane to compute the operations:

Algorithm 7 Parallel Scan (log-space)

Input: $a_{1:L} \in \mathbb{R}^{B \times L \times d}, b_{0:L} \in \mathbb{R}^{B \times (L+1) \times d}$
Output: $h \in \mathbb{R}^{B \times L \times d}$

$\log a \in \mathbb{C}^{B \times L \times d} \leftarrow ComplexLog(a)$
$\log b \in \mathbb{C}^{B \times (L+1) \times d} \leftarrow ComplexLog(b)$
$a^* \in \mathbb{C}^{B \times L \times d} \leftarrow PrefixSum(a)$ ▷ on L axis
$a^* \in \mathbb{C}^{B \times (L+1) \times d} \leftarrow ShiftToRight(a^*)$ ▷ on L axis
$(\log x_0 + b^*) \leftarrow \log PrefixSum\,(\exp(\log b - a^*))$
$\log x \in \mathbb{R}^{B \times (L+1) \times d} \leftarrow \mathrm{Re}(a^* + (\log x_0 + b^*))$
return $\exp(\log x)_{1:L}$

where for any $x \in \mathbb{R}^d$, we have

$$\mathrm{ComplexLog}(x) = \log \max(|x|, \epsilon) + i\theta(x), \quad (9)$$

$$\theta(x) = \begin{cases} 0 & \text{if } x \geq 0, \\ \pi & \text{if } x < 0. \end{cases} \quad (10)$$

3.1 Minimal RNNs

The idea of generating gates based solely on the input was further studied in (Feng et al., 2024) on

[2] https://developer.nvidia.com/gpugems/gpugems3/part-vi-gpu-computing/chapter-39-parallel-prefix-sum-scan-cuda

old RNN architectures. By removing nonlinearities as well, they reduced the architectures to a minimal formulation that retains the essence of SSMs and can be computed efficiently with parallel scan. The minGRU gets rid of the reset gate from GRU (Cho, 2014) so the parameters were reduced down to two, and takes the following form:

Algorithm 8 minGRU (Sequential Mode), $h_0 \leftarrow 0$.

Input: $x \in \mathbb{R}^{B \times L \times d}$
Output: $h \in \mathbb{R}^{B \times L \times d}$
Require: $\text{Linear}_z, \text{Linear}_n$

for $t = 1, 2, \ldots, L$ **do**
  $z_t \leftarrow \sigma(\text{Linear}_z(x_t))$ ▷ update gate
  $n_t \leftarrow \text{Linear}_n(x_t)$ ▷ candidate
  $h_t \leftarrow (1 - z_t) \odot h_{t-1} + z_t \odot n_t$
end for
return $h_{1:L}$

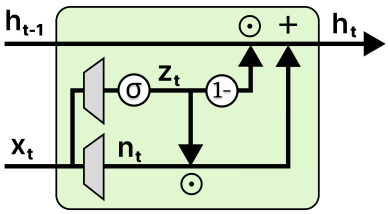


Figure 2: An illustration of minGRU cell (Sequential)

Algorithm 9 minGRU (Parallel Mode), $h_0 \leftarrow 0$.

Input: $x \in \mathbb{R}^{B \times L \times d}$
Output: $h \in \mathbb{R}^{B \times L \times d}$
Require: $\text{Linear}_z, \text{Linear}_n$

$z \leftarrow \sigma(\text{Linear}_z(x))$
$n \leftarrow \text{Linear}_n(x)$
$h \leftarrow ParallelScan((1 - z), [h_0, z \odot n])$
return h

The LSTM (Hochreiter, 1997) conveys a similar reduction as in the minGRU cell, where they removed the hidden state dependency, along with other modifications to ensure the output is time-independent in scale (gate normalization such that the forget and input gate sum up to one and the removal of the output gate along with the cell state).

In their tests on Selective Copying Task and Multilocomotion Reinforcement Learning outperform the baselines (Hyena (Poli et al., 2023), S4) and perform comparably with Mamba and Transformer based models.

Algorithm 10 minLSTM (Sequential Mode), $h_0 \leftarrow 0$.

Input: $x \in \mathbb{R}^{B \times L \times d}$
Output: $h \in \mathbb{R}^{B \times L \times d}$
Require: $\text{Linear}_i, \text{Linear}_f, \text{Linear}_c$

for $t = 1, 2, \ldots, L$ **do**
  $i_t \leftarrow \sigma(\text{Linear}_i(x_t))$ ▷ input gate
  $f_t \leftarrow \sigma(\text{Linear}_f(x_t))$ ▷ forget gate
  $f'_t, i'_t \leftarrow \frac{f_t}{f_t + i_t}, \frac{i_t}{f_t + i_t}$ ▷ norm. gates
  $c_t \leftarrow \text{Linear}_c(x_t)$ ▷ candidate
  $h_t \leftarrow f'_t \odot h_{t-1} + i'_t \odot c_t$
end for
return $h_{1:L}$

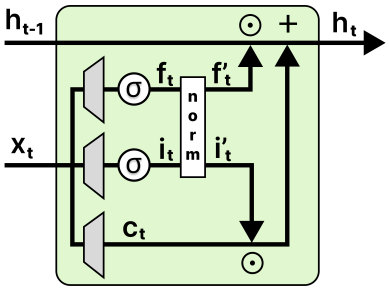


Figure 3: An illustration of minLSTM cell (Sequential)

Algorithm 11 minLSTM (Parallel Mode), $h_0 \leftarrow 0$.

Input: $x \in \mathbb{R}^{B \times L \times d}$
Output: $h \in \mathbb{R}^{B \times L \times d}$
Require: $\text{Linear}_i, \text{Linear}_f, \text{Linear}_c$

$i \leftarrow \sigma(\text{Linear}_i(x))$
$f \leftarrow \sigma(\text{Linear}_f(x))$
$f', i' \leftarrow \frac{f}{f + i}, \frac{i}{f + i}$
$c \leftarrow \text{Linear}_c(x)$
$h \leftarrow ParallelScan(f', [h_0, i' \odot c])$
return $h_{1:L}$

The authors introduced a log-space version for the parallel scan algorithm which ensures numerical stability along with an according implementation for minGRU and minLSTM in the appendix of their paper that leverages it. The parallel mode is used during training to avoid backpropagation through time (BPTT), and the sequential mode is used during inference (considering hidden states caching). Computing the complex logarithm (in log-space parallel scan) requires $\epsilon > 0$ for numerical stability, so the implementations receive certain

modifications that must be followed by checking the original paper.

4 Conclusion

This paper has explored the recent shift toward reconsidering linear sequence modeling approaches as alternatives to attention-based architectures. Through our analysis of Structured State Space Models (S4) and their evolution into Selective State Space Models (S6), we observe a promising direction in achieving efficient sequence processing via parallel scan while maintaining model expressiveness.

While recent architectural innovations such as hybrid approaches (e.g., Zamba (Glorioso et al., 2024), Hymba (Dong et al., 2024)) along other types of arhitectures like Liquid Time-Constant (LTC) Networks (Hasani et al., 2020) based LLMs have demonstrated promising benchmark results, though practical abilities remain susceptible, requiring further investigation.

References

Dzmitry Bahdanau. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Guy E Blelloch. 1990. Prefix sums and their applications.

John S Bridle. 1990. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing: Algorithms, architectures and applications*, pages 227–236. Springer.

Kyunghyun Cho. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Tri Dao and Albert Gu. 2024. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Xin Dong, Yonggan Fu, Shizhe Diao, Wonmin Byeon, Zijia Chen, Ameya Sunil Mahabaleshwarkar, Shih-Yang Liu, Matthijs Van Keirsbilck, Min-Hung Chen, Yoshi Suhara, Yingyan Lin, Jan Kautz, and Pavlo Molchanov. 2024. Hymba: A hybrid-head architecture for small language models.

Leo Feng, Frederick Tung, Mohamed Osama Ahmed, Yoshua Bengio, and Hossein Hajimirsadegh. 2024. Were rnns all we needed? *arXiv preprint arXiv:2410.01201*.

Paolo Glorioso, Quentin Anthony, Yury Tokpanov, James Whittington, Jonathan Pilault, Adam Ibrahim, and Beren Millidge. 2024. Zamba: A compact 7b ssm hybrid model.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee.

Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Albert Gu, Karan Goel, and Christopher Ré. 2022. Efficiently modeling long sequences with structured state spaces.

Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. 2020. Liquid time-constant networks.

Franz A. Heinsen. 2023. Efficient parallelization of a ubiquitous sequential computation.

S Hochreiter. 1997. Long short-term memory. *Neural Computation MIT-Press*.

Rudolf E Kalman. 1960. On the general theory of control systems. In *Proceedings first international conference on automatic control, Moscow, USSR*, pages 481–492.

Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5:341–353.

Eric Martin and Chris Cundy. 2018. Parallelizing linear recurrent neural nets over sequence length.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y. Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. 2023. Hyena hierarchy: Towards larger convolutional language models.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*.

I Sutskever. 2014. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Yuxin Wu and Kaiming He. 2018. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19.