Parallelisable Recurrent Sequence Models

2025-02-06
OccaMLab X OatML Joint Group Seminar

Font = Roboto Mono (normal), title colour = `283618`, comment colour = `6aa84f` (“dark green 1”)

Outline

1. Definitions
2. Papers:
   a. [parallel scan] Blelloch, Guy E. "Prefix sums and their applications." (1990).
   b. Martin, Eric, and Chris Cundy. "Parallelizing linear recurrent neural nets over sequence length." arXiv preprint arXiv:1709.04057 (2017).
   c. Feng, Leo, et al. "Were rnns all we needed?." arXiv preprint arXiv:2410.01201 (2024).
   d. [linear attention] Katharopoulos, Angelos, et al. "Transformers are rnns: Fast autoregressive transformers with linear attention." International conference on machine learning. PMLR, 2020.
   e. [LSSL] Gu, Albert, et al. "Combining recurrent, convolutional, and continuous-time models with linear state space layers." Advances in neural information processing systems 34 (2021): 572-585.
   f. [S4] Gu, Albert, Karan Goel, and Christopher Ré. "Efficiently modeling long sequences with structured state spaces." arXiv preprint arXiv:2111.00396 (2021).
   g. [S5] Smith, Jimmy TH, Andrew Warrington, and Scott W. Linderman. "Simplified state space layers for sequence modeling." arXiv preprint arXiv:2208.04933 (2022).
   h. [Mamba] Gu, Albert, and Tri Dao. "Mamba: Linear-time sequence modeling with selective state spaces." arXiv preprint arXiv:2312.00752 (2023).
   i. [Mamba 2] Dao, Tri, and Albert Gu. "Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality." arXiv preprint arXiv:2405.21060 (2024).
   j. Orvieto, Antonio, et al. "Resurrecting recurrent neural networks for long sequences." International Conference on Machine Learning. PMLR, 2023.
   k. Lu, Chris, et al. "Structured state space models for in-context reinforcement learning." Advances in Neural Information Processing Systems 36 (2024).

Green: stimulus for talk. Found, read, thought cool. Here I am giving seminar. Lots of papers -> interrupt, discuss. Don’t need to cover everything

Definitions

1. Associativity
2. Sequence model
3. Recurrent sequence model
4. Parallelisable sequence model

Associativity = recurring theme. 2 properties of sequence models + motivation. Rattle through !!!

Associativity

- $S = \{s_1, s_2, ...\}$
- Binary operation:
-     $f: S \times S \to S$
-     $f(x, y) = x$
-     $x \circ y = z$

We have a set. Binary operation takes 2 elements, produces 3rd. Dot notation

Associativity

- Associative binary operation ($\forall$ x,y,z):
- $(x \circ y) \circ z$
- $= x \circ (y \circ z)$
- $= x \circ y \circ z$
- -> “Generalized associative law”, EG:
- $((((u \circ v) \circ w) \circ x) \circ y) \circ z$
- $= u \circ (v \circ (w \circ (x \circ (y \circ z))))$
- $= u \circ v \circ w \circ x \circ y \circ z$
- Result $\perp$ order of brackets
- Compute in any order [in time]

We have expression: x dot y dot z. Associative: change order of brackets, **always** same result. ~ don’t write brackets. Generalises to longer expressions (proof: induction). General concept: result independent of order of brackets/computation. Compute products in any order (without changing sequence order)

Associativity

- Examples:
- $x + y$
- $x * y$
- "hello " + "world"
- = "hello world"
- Counterexamples:
- $x - y$
- x / y
- $2(x + y)$

$$(x \circ y) \circ z$$
$$= 2(2x + 2y) + 2z$$
$$= 4x + 4y + 2z$$

$$x \circ (y \circ z)$$
$$= 2x + 2(2y + 2z)$$
$$= 2x + 4y + 4z$$

Addition, multiplication, concatenation (no inverse). Subtraction, division, addition+multiplication (!). Terms “remember” depth (# enclosing brackets)

Sequence model

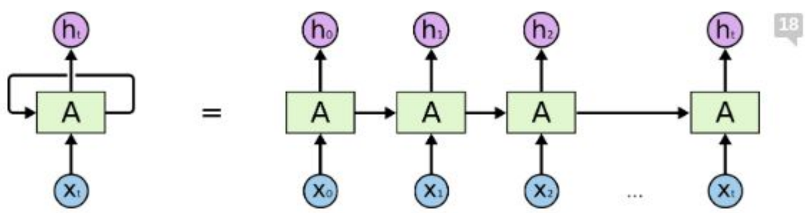
- f:
-     ordered, variable-length input $x_{1:T}$
-     learnable parameters w
- x = text, video, POMDP (RL) states, ...

$$y_t = f(x_1, x_2, ..., x_t; w)$$

Function: input = ordered variable-length sequence, parameters = learnable.
Examples

Recurrent sequence model

- Input -> hidden state -> output
- $h_t = f(x_t, h_{t-1})$
- $y_t = g(h_t)$
- EG RNN (<1997), LSTM (1997), GRU (2014)



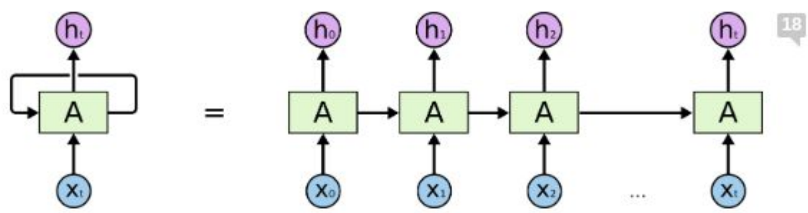
An unrolled recurrent neural network.

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Common theme = hidden state. Update hidden (current input + previous hidden), **discard current input (don't store in memory)**. Output from hidden.

Recurrent sequence model

- Input -> hidden state -> output
- Length N sequence:
- Inference : O(1) space 🙂
- Training : O(N) space 😕 (BPTT)
- Sequential -> hard to parallelise 🙁



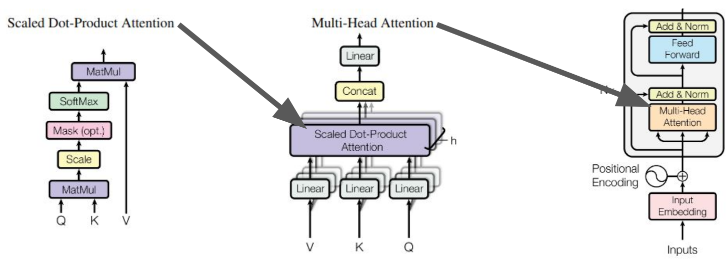
An unrolled recurrent neural network.

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Inference = memory efficient (discard inputs, don't have to store in memory). Training: less memory efficient (compute gradients: backpropagate through computational graph -> store computational graph in memory). Not a problem for humans.
Sequential computation = hard to parallelise -> Slow to train

Parallelisable sequence model

- ⊥ hidden states -> compute in parallel
- Fast training on GPUs* 🙂
- *When data is pre-generated (not in RL 😕)

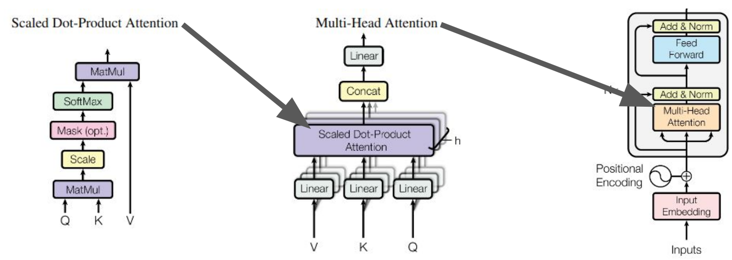


Vaswani, A. "Attention is all you need." Advances in Neural Information Processing Systems (2017).

No dependencies between hidden states, compute in parallel. Fast training on GPUs, EG for language modelling but not for RL

Parallelisable sequence model

- Transformers (2017):
- $q_t, k_t, v_t = f_{qkv}(x_t)$ # Query, key, value
- $w_t = f_w(q_t, k_{1:T})$ # Attention weights
- $a_t = f_a(w_t, v_{1:T})$ # Attention
- $y_t = f_y(x_t, a_t)$ # Output

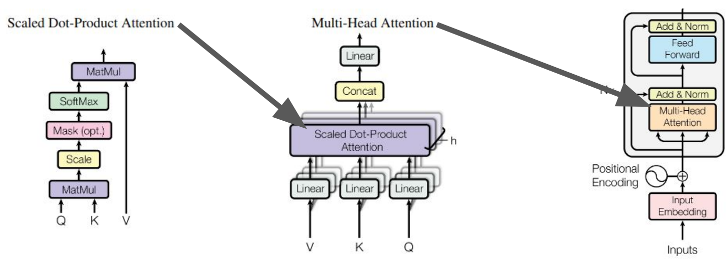


Vaswani, A. "Attention is all you need." Advances in Neural Information Processing Systems (2017).

For each input token: (1) QKV from input (2) attention weight for each query using <u>all keys</u> (3) attention using weights and <u>all values</u> (4) output token from attention and input (residual, PW-MLP, layer norm etc)

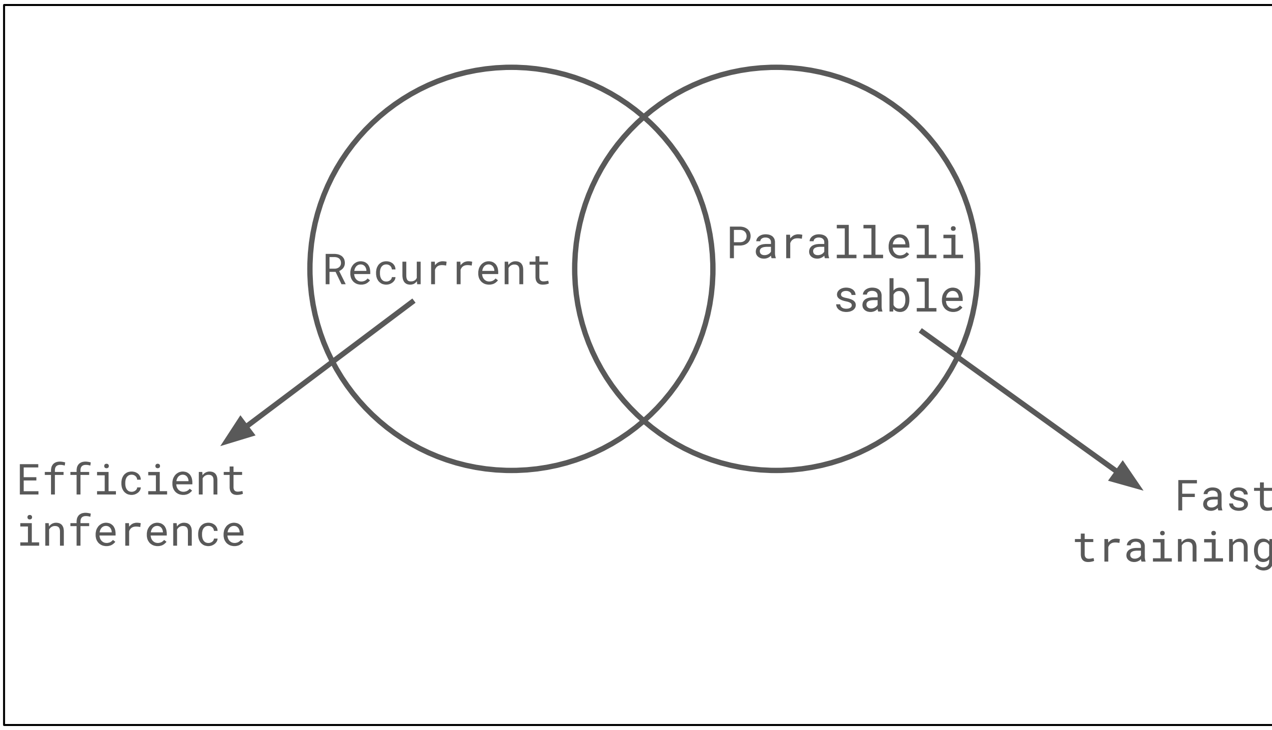
Parallelisable sequence model

- Transformers (2017):
- $a_i \perp a_j$ -> parallelise
- f_w, f_a -> long range dependencies (LRD)

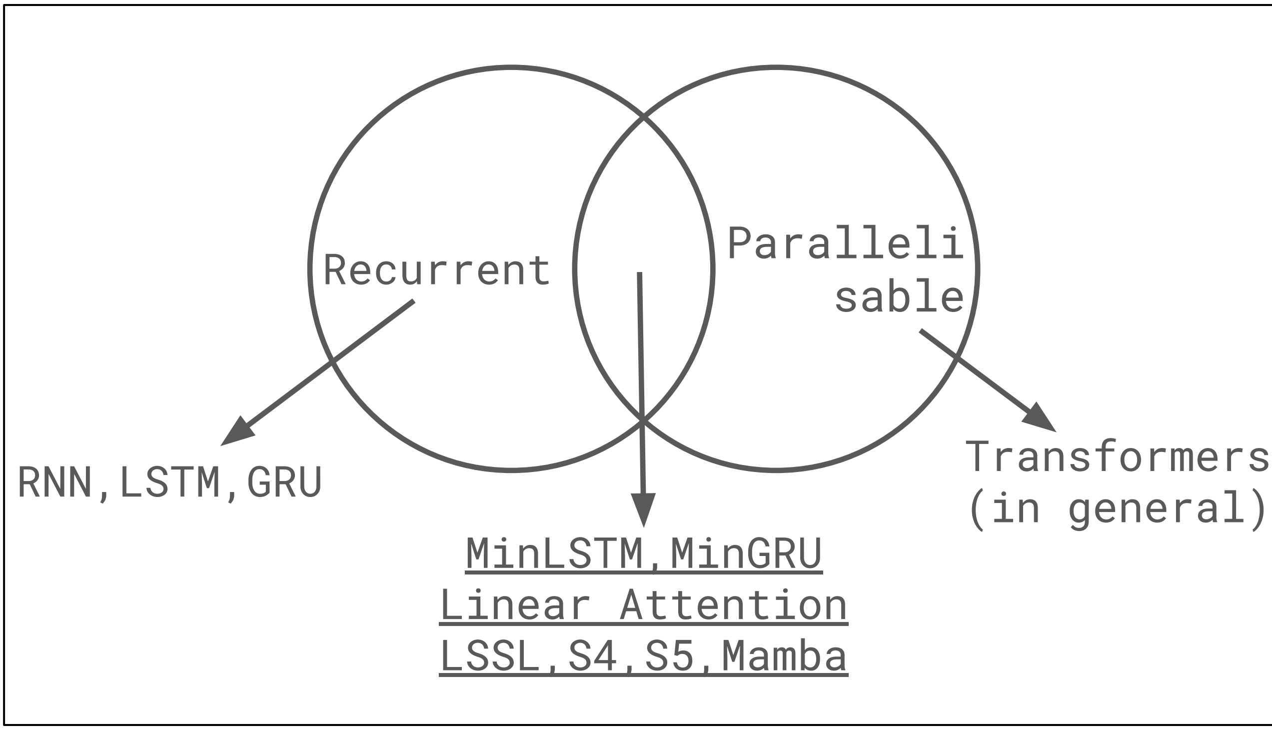


Vaswani, A. "Attention is all you need." Advances in Neural Information Processing Systems (2017).

Attention tokens in a layer = mutually independent -> compute in parallel. Long-range input/output token pairs computed directly (not squashed through many recurrent updates) -> model LRD between distant tokens. Attention (2014): motivation = LRD. Transformer: motivation = attention + parallel



Summarise: recurrent SMs -> efficient inference, parallelisable SMs -> fast training



Focus of talk = intersection

Blelloch, Guy E. "Prefix sums and their applications." (1990).

On to papers. Old paper but …

Blelloch, Guy E. "Prefix sums and their applications." (1990).

Feng, Leo, et al. "Were rnns all we needed?." arXiv preprint arXiv:2410.01201 (2024).

Smith, Jimmy TH, Andrew Warrington, and Scott W. Linderman. "Simplified state space layers for sequence modeling." arXiv preprint arXiv:2208.04933 (2022).

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

… Precursor of many recent papers. Worthwhile to discuss

Prefix sums and their applications

- $\oplus$: *associative binary operator*
- I : *identity of $\oplus$, IE $(\forall a)\ a \oplus I = a$*
- $[a_0, a_1, ..., a_{n-1}]$: *input sequence*

To start off with: assume we have associative binary operator (“plus in circle”/”plus”), identity of the operator (“I”), length-n input sequence (“a”)

Prefix sums and their applications

- `reduce`:
- apply ⊕ to full sequence
- reduce(a) = $a_0 \oplus a_1 \oplus ... \oplus a_{n-1}$

Definition: *The* reduce *operation takes a binary associative operator* $\oplus$ *with identity* i, *and an ordered set* $[a_0, a_1, ..., a_{n-1}]$ *of* n *elements, and returns the value* $a_0 \oplus a_1 \oplus ... \oplus a_{n-1}$.

Define reduce: apply binary operator to full sequence

Prefix sums and their applications

- `scan`:
- scan(a) = [a_0, ($a_0 \oplus a_1$), ..., ($a_0 \oplus a_1 \oplus \ldots \oplus a_{n-1}$)]
- $\mathrm{scan}(a)_i$ = reduce([a_0, ..., a_i])
- $\mathrm{scan}(a)_{n-1}$ = reduce(a)
- CF numerical integration, `numpy.cumsum` etc

For example, if $\oplus$ is addition, then the **scan** operation on the ordered set

$$[3 \quad 1 \quad 7 \quad 0 \quad 4 \quad 1 \quad 6 \quad 3],$$

would return

$$[3 \quad 4 \quad 11 \quad 11 \quad 15 \quad 16 \quad 22 \quad 25].$$

Define scan: ith element of scan = reduce of first i elements of input sequence (first i elements = prefix -> prefix sums). Last element of scan is full reduce. Example

Prefix sums and their applications

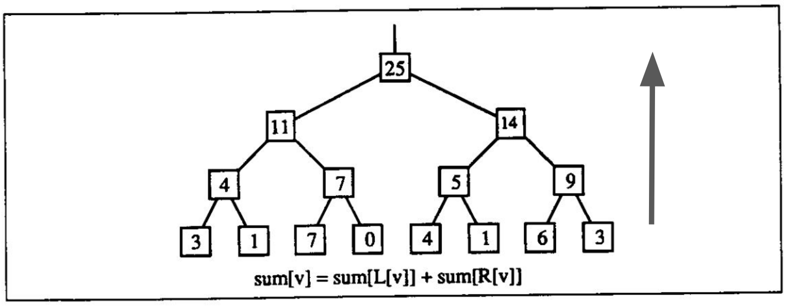
- \`scan\`:
- $\text{scan}(a) = [a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \ldots \oplus a_{n-1})]$
- $\text{scan}(a)_i = \text{reduce}([a_0, \ldots, a_i])$
- $\text{scan}(a)_{n-1} = \text{reduce}(a)$
- CF numerical integration, \`numpy.cumsum\` etc
- Parallel \`scan\` = focus of paper

"... a good example of a computation that seems inherently sequential, but for which there is an efficient parallel algorithm."

Paper = about parallel scan. Interesting because looks sequential: each element can be computed from previous element

Prefix sums and their applications

- Parallel reduce: “up-sweep”



(a) Executing a +-reduce on a tree.

[3 1 7 0 4 1 6 3],

Precursor to parallel scan = parallel **reduce**. Because operator = associative -> compute operations in any order. Parallel hardware -> most efficient = tree. Computation from leaves to root -> “up-sweep”

Prefix sums and their applications

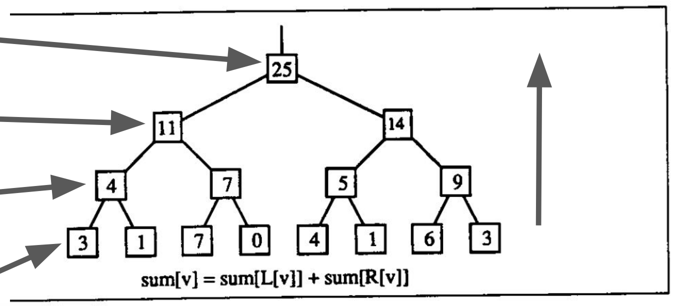
- Parallel reduce: “up-sweep”
- Up-sweep -> (partial) scan results:

$25 = \text{scan}(a)_7$

$11 = \text{scan}(a)_3$

$4 = \text{scan}(a)_1$

$3 = \text{scan}(a)_0$



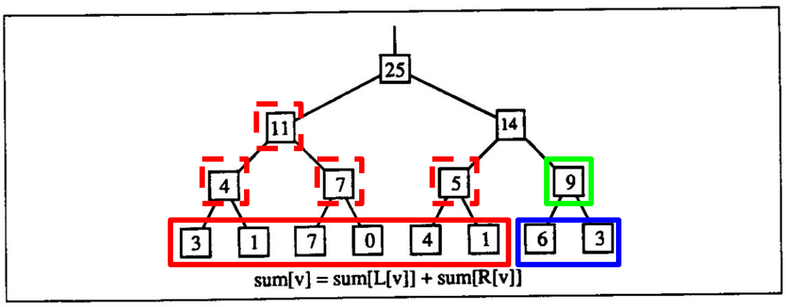
(a) Executing a +-reduce on a tree.

[3 1 7 0 4 1 6 3],

Computation tree contains partial sums. Partial sums: (partial) scan results. Question: how to use these to compute scan?

Prefix sums and their applications

- Parallel scan:
- Up-sweep : sum(all descendent leaves)
- Down-sweep : sum(all preceding leaves)
- “preceding” = left of all descendent leaves

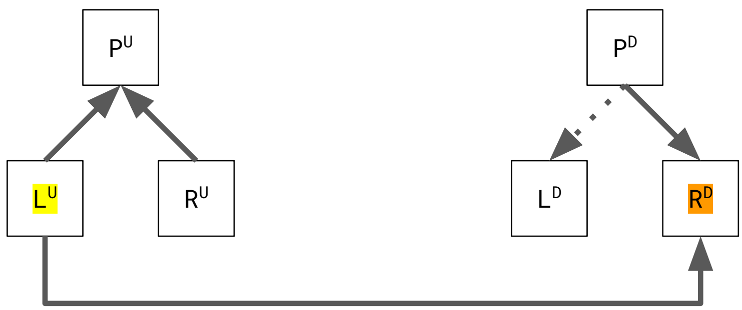


(a) Executing a +-reduce on a tree.

Parallel scan intuition: up sweep + **down sweep**, node values = sums

Prefix sums and their applications

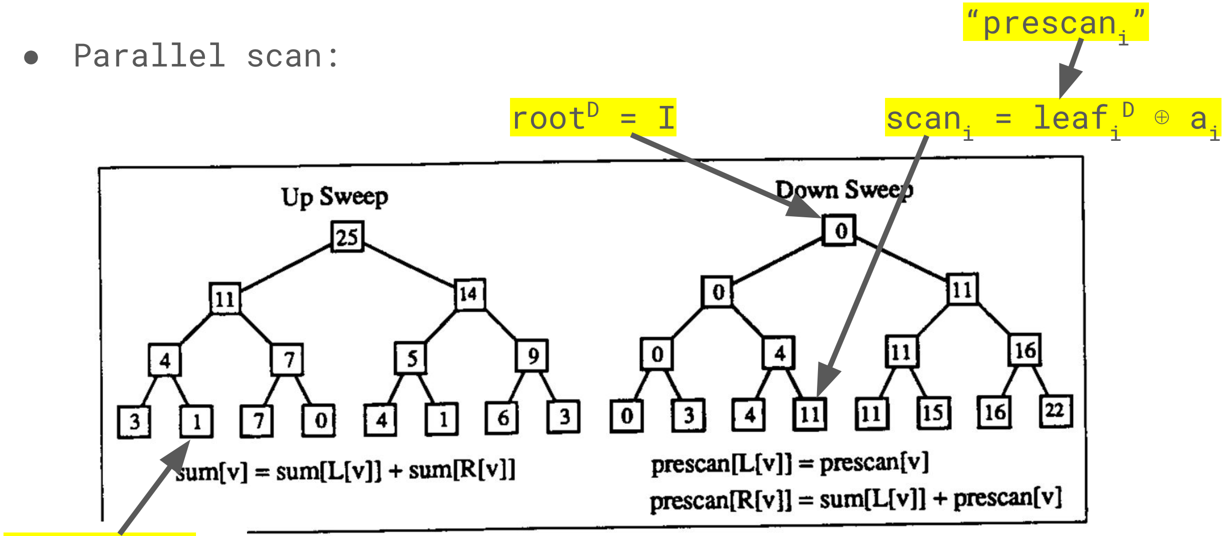
- Parallel scan:
- Up-sweep : $P^U = L^U \oplus R^U$ # *Parent : sum of children*
- Down-sweep : $L^D = P^D$ # *Left : copy parent*
- $R^D = P^D \oplus L^U$ # *Right : parent + left-US*



Consider 3 nodes: L (left child), R (right child), P (parent). Assume P^D already has sum of all preceding leaves. Preceding leaves of L = preceding leaves of P. So $L^D = P^D$. Preceding leaves of R = preceding leaves of P **+ descendent leaves of L**. Sum of descendent leaves of L = L^U (from up-sweep). So $R^D = P^D + L^U$. Preceding leaves of root = empty set, so initialise root = identity on down-sweep. Solid arrows: sum. Dashed arrow: copy

Prefix sums and their applications

- Parallel scan:



(a) Executing a +-prescan on a tree.

Up sweep: initialise leaves to input sequence. Down sweep: initialise root node to identity. Scan(a) = down-sweep leaf values + input sequence (leaf values = “prescan”). Proof in paper

Prefix sums and their applications

- Parallel scan:

procedure down-sweep(A)
  $a[n-1] \leftarrow 0$
  for d **from** $(\lg n) - 1$ **downto** 0
    in parallel for i **from** 0 **to** $n-1$ **by** 2^{d+1}
      $t \leftarrow a[i + 2^d - 1]$ `% Save in temporary`
      $a[i + 2^d - 1] \leftarrow a[i + 2^{d+1} - 1]$ `% Set left child`
      $a[i + 2^{d+1} - 1] \leftarrow t + a[i + 2^{d+1} - 1]$ `% Set right child`

	Step	Array in Memory							
	0	$\boxed{3}$	$\boxed{1}$	$\boxed{7}$	$\boxed{0}$	$\boxed{4}$	$\boxed{1}$	$\boxed{6}$	$\boxed{3}$
up	1	3	$\boxed{4}$	7	$\boxed{7}$	4	$\boxed{5}$	6	$\boxed{9}$
	2	3	4	7	$\boxed{11}$	4	5	6	$\boxed{14}$
	3	3	4	7	11	4	5	6	$\boxed{25}$
clear	4	3	4	7	11	4	5	6	$\boxed{0}$
down	5	3	4	7	$\boxed{0}$	4	5	6	$\boxed{11}$
	6	3	$\boxed{0}$	7	$\boxed{4}$	4	$\boxed{11}$	6	$\boxed{16}$
	7	$\boxed{0}$	$\boxed{3}$	$\boxed{4}$	$\boxed{11}$	$\boxed{11}$	$\boxed{15}$	$\boxed{16}$	$\boxed{22}$

(b) Executing a +-prescan on a P-RAM.

Figure 4: A parallel prescan on a tree using integer addition as the associative operator $\oplus$.

Demonstrate implementation with not much additional memory

Prefix sums and their applications

- Scan = **recurrence**:
- $x_i = a_0 \oplus a_1 \oplus \ldots \oplus a_i$
- $x_i = \{ a_0 \quad i = 0$
- $\{ x_{i-1} \oplus a_i \quad 0 < i < n$
- Also parallelisable:
- $x_i = \{ b_0 \quad i = 0$
- $\{ a_i x_{i-1} + b_i \quad 0 < i < n$
- Proof sketch...

Last thing to say about parallel scan = expresses **recurrence**. Can compute any linear recurrence as parallel scan

Prefix sums and their applications

- $x_i = a_i x_{i-1} + b_i$
- $= a_i(a_{i-1}x_{i-2} + b_{i-1}) + b_i$ *# Expand x_{i-1}*
- $= (a_i a_{i-1})x_{i-2} + (a_i b_{i-1} + b_i)$ *# Collect*
- Define:
- $c_i = [a_i, b_i]$
- $c_j \bullet c_i = [a_i a_j, a_i b_j + b_i]$ *# • is associative*
- *# (proof in paper)*
- $s_{i-1} = [A_{i-1}, x_{i-1}]$
- $A_i = a_i A_{i-1}$
- $\Rightarrow s_{i-1} \bullet c_i = [a_i A_{i-1}, a_i x_{i-1} + b_i]$
- $= [A_i, x_i]$
- $= s_i$ *# "scan" recurrence*

Proof sketch: introduce new associative operator on **2D vectors**, derive "scan" recurrence that contains original recurrence in one of the dimensions. NB coefficients can be different on every time step (EG function of input data)

Martin, Eric, and Chris Cundy. "Parallelizing linear recurrent neural nets over sequence length." arXiv preprint arXiv:1709.04057 (2017).

Martin, 2017

- Recognise connection RNN : parallel scan (!)
- Classify parallelisable RNNs, EG QRNN [1]
- Implement parallel scan CUDA kernel, 9x speed-ups
- Linear recurrence:
- Only linear within each layer
- Stack multiple layers + nonlinearities
- ⇒ Nonlinear dependence on past input tokens

[1] Bradbury, James, et al. "Quasi-recurrent neural networks." arXiv preprint arXiv:1611.01576 (2016).

Martin, 2017

- Introduce GILR: linear recurrence + nonlinear gating

3.1 Gated impulse linear recurrent layer

A gated impulse linear recurrent (GILR) layer transforms its m dimensional inputs x_t into a sequence of n dimensional hidden states h_t:

$$g_t = \sigma(Ux_t + b_g)$$
$$i_t = \tau(Vx_t + b_z)$$
$$h_t = g_t \odot h_{t-1} + (1 - g_t) \odot i_t$$

A GILR layer applies the same non-linear transform to each sequence element and then accumulates the sequence elements with a non-linear gating mechanism. Gate g_t uses the sigmoid activation function to give values in [0,1] for reasonable gating semantics, while impulse i_t can use any activation function τ. Stacking GILR layers allows for rich non-linear dependence on previous events while still taking advantage of fast parallel sequence evaluation.

Martin, 2017

- Results >> LSTM (limited experiments)

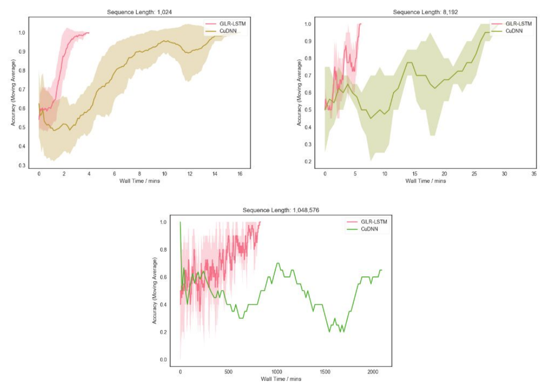


Figure 2: Learning curves for GILR-LSTM and CuDNN LSTM architectures for various sequence lengths. Each plot shows the moving mean and standard deviation of classification accuracy over five training runs, with the exception of a single run for CuDNN LSTM on 1 million sequence length.

Feng, Leo, et al. "Were rnns all we needed?." arXiv preprint arXiv:2410.01201 (2024).

Were rnns all we needed?

- Derive simplified LSTM/GRU (“MinLSTM/MinGRU”):
-     (1) Fewer parameters
-     (2) Parallelisable training
-     (3) “surprisingly competitive performance” (abstract)
- NB only MinLSTM vs MinGRU difference:
-     forget <u>and</u> input gates (MinLSTM)
-     single gate (MinGRU)

Were rnns all we needed?

LSTM

$$\begin{aligned}
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t) \\
\boldsymbol{o}_t &= \sigma(\text{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}])) \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \\
\boldsymbol{f}_t &= \sigma(\text{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}])) \\
\boldsymbol{i}_t &= \sigma(\text{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}])) \\
\tilde{\boldsymbol{c}}_t &= \tanh(\text{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}]))
\end{aligned}$$

$$\Rightarrow$$

minLSTM

$$\begin{aligned}
\boldsymbol{h}_t &= \boldsymbol{f}_t \odot \boldsymbol{h}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{h}}_t \\
\boldsymbol{f}_t &= \sigma(\text{Linear}_{d_h}(\boldsymbol{x}_t)) \\
\boldsymbol{i}_t &= \sigma(\text{Linear}_{d_h}(\boldsymbol{x}_t)) \\
\tilde{\boldsymbol{h}}_t &= \text{Linear}_{d_h}(\boldsymbol{x}_t)
\end{aligned}$$

Were rnns all we needed?

GRU

$$
\begin{aligned}
\boldsymbol{h}_t &= (\mathbf{1} - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t \\
\boldsymbol{z}_t &= \sigma(\mathrm{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}])) \\
\boldsymbol{r}_t &= \sigma(\mathrm{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{h}_{t-1}])) \\
\tilde{\boldsymbol{h}}_t &= \tanh(\mathrm{Linear}_{d_h}([\boldsymbol{x}_t, \boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}]))
\end{aligned}
$$

$\Rightarrow$

minGRU

$$
\begin{aligned}
\boldsymbol{h}_t &= (\mathbf{1} - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t \\
\boldsymbol{z}_t &= \sigma(\mathrm{Linear}_{d_h}(\boldsymbol{x}_t)) \\
\tilde{\boldsymbol{h}}_t &= \mathrm{Linear}_{d_h}(\boldsymbol{x}_t)
\end{aligned}
$$

You might think MinGRU looks familiar…

Were rnns all we needed?

- “Notably, minGRU is equivalent to GILR but without an activation function” ... !?

$$g_t = \sigma(Ux_t + b_g)$$
$$i_t = \tau(Vx_t + b_z)$$
$$h_t = g_t \odot h_{t-1} + (1 - g_t) \odot i_t$$

minGRU

$$\boldsymbol{h}_t = (\boldsymbol{1} - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t$$
$$\boldsymbol{z}_t = \sigma(\mathrm{Linear}_{d_h}(\boldsymbol{x}_t))$$
$$\tilde{\boldsymbol{h}}_t = \mathrm{Linear}_{d_h}(\boldsymbol{x}_t)$$

“GILR” Martin, Eric, and Chris Cundy. "Parallelizing linear recurrent neural nets over sequence length." arXiv preprint arXiv:1709.04057 (2017).

“minGRU” Feng, Leo, et al. "Were rnns all we needed?." arXiv preprint arXiv:2410.01201 (2024).

MinGRU = rip off of GILR in 2017

Were rnns all we needed?

- Linear recurrence/expressivity:
- 1st layer gates
- = f(current input)
- ≠ f(previous input)
- 1st layer output = f(previous input)
- -> 2+ layer gates = f(previous input)

Model	# Layers	Accuracy
MinLSTM	1	37.6 ± 2.0
	2	85.7 ± 5.8
	3	96.0 ± 2.8
MinGRU	1	37.0 ± 2.3
	2	96.8 ± 3.2
	3	99.5 ± 0.2

Table 1: Comparison of the number of layers on the Selective Copying Task (Gu & Dao, 2024).

First layer gates have limited expressivity. Deeper layer gates are more expressive. Reflected in results

Were rnns all we needed?

- Selective copy + RL results: reasonable

Model	Layer	Accuracy
H3	Hyena	30.1
Mamba	Hyena	28.4
S4	S4	18.3
H3	S4	57.0
Mamba	S4	56.4
S4	S6	97.0
H3	S6	99.7
Mamba	S6	99.8
minGRU	minGRU	99.5 ± 0.2
minLSTM	minLSTM	96.0 ± 2.8

Table 2: Selective Copy Task. minLSTM, minGRU, and Mamba's S6 (Gu & Dao, 2024) are capable of solving this task. Other methods such as S4, H3, and Hyena at best only partially solve the task.

Dataset	DT	DS4	DAaren	DMamba	minLSTM	minGRU
HalfCheetah-M	42.6	42.5	42.2	42.8	42.7 ± 0.7	43.0 ± 0.4
Hopper-M	68.4	54.2	80.9	83.5	85.0 ± 4.4	79.4 ± 8.2
Walker-M	75.5	78.0	74.4	78.2	72.0 ± 7.5	73.3 ± 3.3
HalfCheetah-M-R	37.0	15.2	37.9	39.6	38.6 ± 1.1	38.5 ± 1.1
Hopper-M-R	85.6	49.6	77.9	82.6	88.5 ± 4.7	90.5 ± 0.9
Walker-M-R	71.2	69.0	71.4	70.9	69.7 ± 10.7	72.8 ± 8.9
HalfCheetah-M-E	88.8	92.7	75.7	91.9	85.4 ± 1.7	86.3 ± 0.5
Hopper-M-E	109.6	110.8	103.9	111.1	110.3 ± 1.6	109.7 ± 2.7
Walker-M-E	109.3	105.7	110.5	108.3	110.3 ± 0.5	110.3 ± 0.4
Average	76.4	68.6	75.0	78.8	78.1	78.2

Table 3: Reinforcement Learning results on the D4RL (Fu et al., 2020) datasets. We report the expert normalized returns (higher is better), following (Fu et al., 2020), averaged across five random seeds. The minimal versions of LSTM and GRU, minLSTM and minGRU outperform Decision S4 (David et al., 2023) and perform comparably with Decision Mamba (Ota, 2024), (Decision) Aaren (Feng et al., 2024) and Decision Transformer (Chen et al., 2021).

Katharopoulos, Angelos, et al. "Transformers are rnns: Fast autoregressive transformers with linear attention." International conference on machine learning. PMLR, 2020.

Linear attention

- Self-attention = linear dot-product of kernel feature maps (not softmax)
- Associativity: $O(N^2)$ -> $O(N)$
- “Our linear transformers achieve similar performance to vanilla transformers and they are up to 4000x faster on autoregressive prediction of very long sequences”

Linear attention

- Softmax attention:

$$
\begin{aligned}
Q &= xW_Q, \\
K &= xW_K, \\
V &= xW_V, \\
A_l(x) = V' &= \operatorname{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)V.
\end{aligned}
$$

Linear attention

- “Similarity” attention (generalisation):

$$V_i' = \frac{\sum_{j=1}^{N} \mathrm{sim}\left(Q_i, K_j\right) V_j}{\sum_{j=1}^{N} \mathrm{sim}\left(Q_i, K_j\right)}. \tag{3}$$

Equation 3 is equivalent to equation 2 if we substitute the similarity function with $\mathrm{sim}\left(q, k\right) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$.

Linear attention

- “Kernel” attention (separable similarity):

Given such a kernel with a feature representation $\phi(x)$ we can rewrite equation 2 as follows,

$$V_i' = \frac{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j) V_j}{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j)}, \tag{4}$$

and then further simplify it by making use of the associative property of matrix multiplication to

$$V_i' = \frac{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j)}. \tag{5}$$

Linear attention

- “Transformers are RNNs”

Associative,
recurrent ->
parallelisable
(parallel scan)

Accumulate
attention
memory

Accumulate
“normaliser”
memory

Compute output

$$s_0 = 0, \tag{16}$$

$$z_0 = 0, \tag{17}$$

$$s_i = s_{i-1} + \phi\left(x_i W_K\right)\left(x_i W_V\right)^T, \tag{18}$$

$$z_i = z_{i-1} + \phi\left(x_i W_K\right), \tag{19}$$

$$y_i = f_l\left(\frac{\phi\left(x_i W_Q\right)^T s_i}{\phi\left(x_i W_Q\right)^T z_i} + x_i\right). \tag{20}$$

Linear attention

- Results: speed = great, performance = reasonable (limited comparisons)

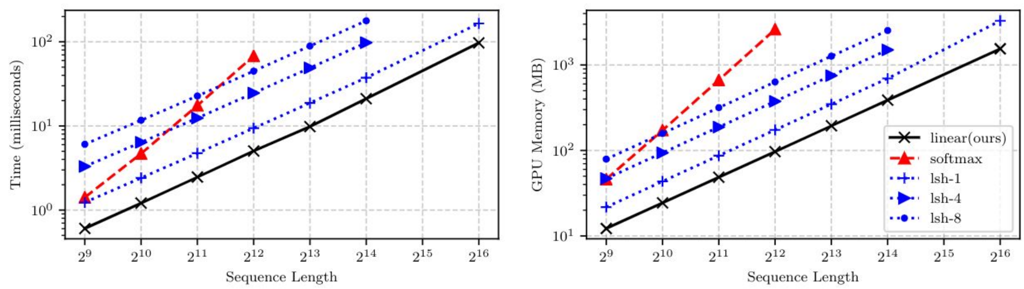


Figure 1: Comparison of the computational requirements for a forward/backward pass for Reformer (lsh-X), softmax attention and linear attention. Linear and Reformer models scale linearly with the sequence length unlike softmax which scales with the square of the sequence length both in memory and time. Full details of the experiment can be found in § 4.1.

Linear attention

- Results: speed = great, performance = reasonable (limited comparisons)

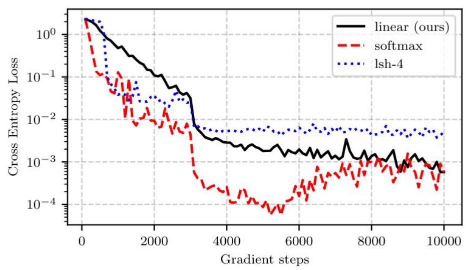


Figure 2: Convergence comparison of *softmax*, *linear* and *reformer* attention on a sequence duplication task. *linear* converges stably and reaches the same final performance as softmax. The details of the experiment are in § 4.1.

Method	Validation PER	Time/epoch (s)
Bi-LSTM	10.94	1047
Softmax	5.12	2711
LSH-4	9.33	2250
Linear (ours)	8.08	**824**

Table 3: Performance comparison in automatic speech recognition on the WSJ dataset. The results are given in the form of phoneme error rate (PER) and training time per epoch. Our model outperforms the LSTM and Reformer while being faster to train and evaluate. Details of the experiment can be found in § 4.3.

Linear attention

- Results: speed = great, performance = reasonable (limited comparisons)

Method	Bits/dim	Images/sec	
Softmax	0.621	0.45	($1\times$)
LSH-1	0.745	0.68	($1.5\times$)
LSH-4	0.676	0.27	($0.6\times$)
Linear (ours)	0.644	**142.8**	**($317\times$)**

Table 1: Comparison of autoregressive image generation of MNIST images. Our linear transformers achieve almost the same bits/dim as the full softmax attention but more than 300 times higher throughput in image generation. The full details of the experiment are in § 4.2.1.

Method	Bits/dim	Images/sec	
Softmax	3.47	0.004	($1\times$)
LSH-1	3.39	0.015	($3.75\times$)
LSH-4	3.51	0.005	($1.25\times$)
Linear (ours)	3.40	**17.85**	**($4{,}462\times$)**

Table 2: We train autoregressive transformers for 1 week on a single GPU to generate CIFAR-10 images. Our linear transformer completes 3 times more epochs than softmax, which results in better perplexity. Our model generates images $4{,}000\times$ faster than the baselines. The full details of the experiment are in § 4.2.2.

Gu, Albert, et al. "Combining recurrent, convolutional, and continuous-time models with linear state space layers." Advances in neural information processing systems 34 (2021): 572-585.

Linear state space layers

- Stacked layers of linear state space models
- Position-wise nonlinearities between layers
- Well established theory
-     EG (input : output) ~ convolution (impulse response)

Our first goal is to construct an expressive model family that combines all 3 paradigms while preserving their strengths. The **Linear State-Space Layer (LSSL)** is a simple sequence model that maps a 1-dimensional function or sequence $u(t) \mapsto y(t)$ through an implicit state $x(t)$ by simulating a linear continuous-time state-space representation in discrete-time

$$\dot{x}(t) = Ax(t) + Bu(t) \quad (1)$$
$$y(t) = Cx(t) + Du(t), \quad (2)$$

where A controls the evolution of the system and B, C, D are projection parameters. The LSSL can be viewed as an instantiation of each family, inheriting their strengths (Fig. 1):

- **LSSLs are recurrent.** If a discrete step-size Δt is specified, the LSSL can be discretized into a linear recurrence using standard techniques, and simulated during inference as a stateful recurrent model with constant memory and computation per time step.
- **LSSLs are convolutional.** The linear time-invariant systems defined by (1)+(2) are known to be explicitly representable as a continuous convolution. Moreover, the discrete-time version can be parallelized during training using convolutions [12, 44].
- **LSSLs are continuous-time.** The LSSL itself is a differential equation. As such, it can perform unique applications of continuous-time models, such as simulating continuous processes, handling missing data [45], and adapting to different timescales.

SSMs have much established theory, including “impulse response”

Linear state space layers

- Stacked layers of linear state space models
- Generalises RNNs + CNNs
- Preserve information in LRDs ("continuous time memorization")

Summary of Contributions

- We introduce Linear State-Space Layers (LSSLs), a simple sequence-to-sequence transformation that shares the modeling advantages of recurrent, convolutional, and continuous-time methods. Conversely, we show that RNNs and CNNs can be seen as special cases of LSSLs (Section 3).
- We prove that a structured subclass of LSSLs can learn representations that solve continuous-time memorization, allowing it to adapt its measure and timescale (Section 4.1). We also provide new algorithms for these LSSLs, showing that they can be sped up computationally under an arithmetic complexity model Section 4.2.
- Empirically, we show that LSSLs stacked into a deep neural network are widely effective on time series data, even (or *especially*) on extremely long sequences (Section 5).

Linear state space layers

- Discrete approximation for continuous linear SSM
- Recurrence or convolution (can parallelise with FFT)

$$x_t = \overline{A}x_{t-1} + \overline{B}u_t$$
$$y_t = Cx_t + Du_t.$$

As a recurrence. The recurrent state $x_{t-1} \in \mathbb{R}^{H\times N}$ carries the context of all inputs before time t. The current state x_t and output y_t can be computed by simply following equations (4)+(5). Thus the LSSL is a recurrent model with efficient and stateful inference, which can consume a (potentially unbounded) sequence of inputs while requiring fixed computation/storage per time step.

As a convolution. For simplicity let the initial state be $x_{-1} = 0$. Then (4)+(5) explicitly yields

$$y_k = C\left(\overline{A}\right)^k \overline{B}u_0 + C\left(\overline{A}\right)^{k-1}\overline{B}u_1 + \cdots + C\overline{A}\overline{B}u_{k-1} + \overline{B}u_k + Du_k. \tag{6}$$

Then y is simply the (non-circular) convolution $y = \mathcal{K}_L(\overline{A}, \overline{B}, C) * u + Du$, where

$$\mathcal{K}_L(A, B, C) = \left(CA^iB\right)_{i\in[L]} \in \mathbb{R}^L = (CB, CAB, \ldots, CA^{L-1}B). \tag{7}$$

Thus the LSSL can be viewed as a convolutional model where the entire output $y \in \mathbb{R}^{H\times L}$ can be computed at once by a convolution, which can be efficiently implemented with three FFTs.

Connection of FFT vs DFT to parallel scan?

Linear state space layers

- HIPPO theory: choose `A` (closed form) to provably memorise LRD

The **translated Legendre (LegT)** measures assign uniform weight to the most recent history $[t-\theta, t]$. There is a hyperparameter θ representing the length of the sliding window, or the length of history that is being summarized. The **translated Laguerre (LagT)** measures instead use the exponentially decaying measure, assigning more importance to recent history.

$$\textbf{LegT}: \mu^{(t)}(x) = \frac{1}{\theta}\mathbb{I}_{[t-\theta,t]}(x) \qquad \textbf{LagT}: \mu^{(t)}(x) = e^{-(t-x)}\mathbb{I}_{(-\infty,t]}(x) = \begin{cases} e^{x-t} & \text{if } x \le t \\ 0 & \text{if } x > t \end{cases}$$

Theorem 1. *For LegT and LagT, the* hippo *operators satisfying Definition 1 are given by linear time-invariant (LTI) ODEs* $\frac{d}{dt}c(t) = -Ac(t) + Bf(t)$, *where* $A \in \mathbb{R}^{N\times N}, B \in \mathbb{R}^{N\times 1}$:

LegT:

$$A_{nk} = \frac{1}{\theta}\begin{cases} (-1)^{n-k}(2n+1) & if\, n \ge k \\ 2n+1 & if\, n \le k \end{cases}, \quad B_n = \frac{1}{\theta}(2n+1)(-1)^n \tag{1}$$

LagT:

$$A_{nk} = \begin{cases} 1 & if\, n \ge k \\ 0 & if\, n < k \end{cases}, \quad B_n = 1 \tag{2}$$

Gu, Albert, et al. "Hippo: Recurrent memory with optimal polynomial projections." Advances in neural information processing systems 33 (2020): 1474-1487.

Connection of FFT vs DFT to parallel scan?

Linear state space layers

- Strong results (selected problems)
- s/p = sequential/permuted
- Limitation: space complexity

Table 1: **(Pixel-by-pixel image classification.)** (Top) our methods. (Middle) recurrent baselines. (Bottom) convolutional + other baselines.

Model	sMNIST	pMNIST	sCIFAR
LSSL	**99.53**	**98.76**	**84.65**
LSSL-fixed	**99.50**	**98.60**	**81.97**
LipschitzRNN	99.4	96.3	64.2
LMUFFT [12]	-	98.49	-
UNIcoRNN [47]	-	98.4	-
HiPPO-RNN [24]	98.9	98.3	61.1
URGRU [25]	99.27	96.51	74.4
IndRNN [34]	99.0	96.0	-
Dilated RNN [8]	98.0	96.1	-
r-LSTM [56]	98.4	95.2	72.2
CKConv [44]	99.32	98.54	63.74
TrellisNet [4]	99.20	98.13	73.42
TCN [3]	99.0	97.2	-
Transformer [56]	98.9	97.9	62.2

Table 2: **(Vital signs prediction.)** RMSE for predicting respiratory rate (RR), heart rate (HR), and blood oxygen (SpO2). * indicates our own runs to complete results for the strongest baselines.

Model	RR	HR	SpO2
LSSL	**0.350**	**0.432**	**0.141**
LSSL-fixed	**0.378**	**0.561**	**0.221**
UnICORNN [47]	1.06	1.39	0.869*
coRNN [47]	1.45	1.81	-
CKConv	1.214*	2.05*	1.051*
NRDE [37]	1.49	2.97	1.29
IndRNN [47]	1.47	2.1	-
expRNN [47]	1.57	1.87	-
LSTM	2.28	10.7	-
Transformer	2.61*	12.2*	3.02*
XGBoost [55]	1.67	4.72	1.52
Random Forest [55]	1.85	5.69	1.74
Ridge Regress. [55]	3.86	17.3	4.16

Connection of FFT vs DFT to parallel scan?

Gu, Albert, Karan Goel, and Christopher Ré. "Efficiently modeling long sequences with structured state spaces." arXiv preprint arXiv:2111.00396 (2021).

S4

- LSSL: high space complexity
- S4: efficient reparameterisation of SSM
-     Condition `A` with low-rank correction
-     -> Stable diagonalisation
- Strong results:
-     “SoTA on every task from the Long Range Arena”
-     “as efficient as all competitors”
-     “closing the gap to Transformers... performing generation 60× faster”

S4

- Parameterise \`A\` as NPLR
- Normal: commutes with transpose
- EG orthogonal, symmetric
- -> Efficient computation using Woodbury identity

Our techniques apply to any matrix that can be decomposed as ***Normal Plus Low-Rank (NPLR)***.

Theorem 1. *All HiPPO matrices from [16] have a NPLR representation*

$$\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^* - \boldsymbol{P}\boldsymbol{Q}^\top = \boldsymbol{V}\left(\boldsymbol{\Lambda} - (\boldsymbol{V}^*\boldsymbol{P})(\boldsymbol{V}^*\boldsymbol{Q})^*\right)\boldsymbol{V}^* \tag{6}$$

for unitary $\boldsymbol{V} \in \mathbb{C}^{N \times N}$*, diagonal* $\boldsymbol{\Lambda}$*, and low-rank factorization* $\boldsymbol{P}, \boldsymbol{Q} \in \mathbb{R}^{N \times r}$*. These matrices HiPPO- LegS, LegT, LagT all satisfy* $r = 1$ *or* $r = 2$*. In particular, equation (2) is NPLR with* $r = 1$*.*

S4

- Results:

Table 4: (**Long Range Arena**) (*Top*) Original Transformer variants in LRA. Full results in Appendix D.2. (*Bottom*) Other models reported in the literature. *Please read Appendix D.5 before citing this table.*

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Transformer	36.37	64.27	57.46	42.44	71.40	✗	53.66
Reformer	<u>37.27</u>	56.10	53.40	38.07	68.50	✗	50.56
BigBird	36.05	64.02	59.29	40.83	74.87	✗	54.17
Linear Trans.	16.13	<u>65.90</u>	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	<u>77.80</u>	✗	54.42
Nyströmformer	37.15	65.52	<u>79.56</u>	41.58	70.94	✗	57.46
Luna-256	37.25	64.57	79.29	<u>47.38</u>	77.72	✗	<u>59.37</u>
S4	**59.60**	**86.82**	**90.90**	**88.65**	**94.20**	**96.35**	**86.09**

S4

- Results:

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**×)

S4

- HIPPO initialisation = important

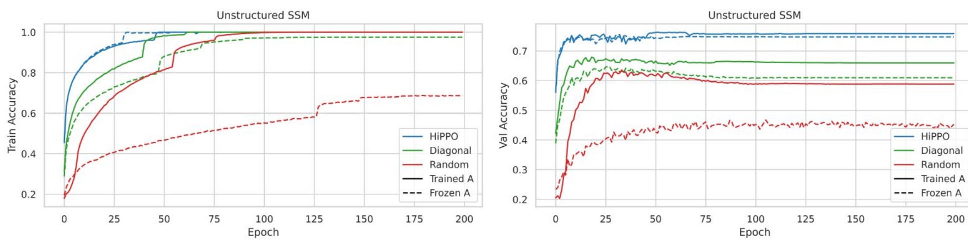


Figure 3: CIFAR-10 classification with unconstrained, real-valued SSMs with various initializations. (*Left*) Train accuracy. (*Right*) Validation accuracy.

Smith, Jimmy TH, Andrew Warrington, and Scott W. Linderman. "Simplified state space layers for sequence modeling." arXiv preprint arXiv:2208.04933 (2022).

S5

- Smith, Jimmy TH, Andrew Warrington, and Scott W. Linderman. "Simplified state space layers for sequence modeling." arXiv preprint arXiv:2208.04933 (2022).
  - Replace many independent SISO SSMs (S4) with one MIMO SSM
  - Train with **parallel scan**
  - “match the computational efficiency of S4, while also achieving state-of-the-art performance on several long-range sequence modeling tasks”

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

MAMBA

- Gu, Albert, and Tri Dao. "Mamba: Linear-time sequence modeling with selective state spaces." arXiv preprint arXiv:2312.00752 (2023).
- SSM params = f(input)
- Train with parallel scan + CUDA kernel fusion
- No attention/MLP blocks
- “On language modeling, our Mamba-3B model outperforms Transformers of the same size and matches Transformers twice its size”

Dao, Tri, and Albert Gu. "Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality." arXiv preprint arXiv:2405.21060 (2024).

Orvieto, Antonio, et al. "Resurrecting recurrent neural networks for long sequences." International Conference on Machine Learning. PMLR, 2023.

Lu, Chris, et al. "Structured state space models for in-context reinforcement learning." Advances in Neural Information Processing Systems 36 (2024).