Signal Processing Formulations of Sequence Models

Julius Smith

DSP Online Conference

www.dsponlineconference.com

October 29-31, 2024

Overview

- Sequence Models, such as Large Language Models (LLMs), arose from a blend of principle-based design and empirical discovery, spanning several fields.
- This talk describes how the ideas could have emerged from an elementary signal-processing approach.
- This viewpoint offers some features:
  1. Signal processing folks can quickly learn what is happening in a motivated way
  2. Machine-learning experts might benefit from signal-processing insights
  3. Obvious suggestions for things to try next naturally arise

Plan: “Invent” the components of modern sequence models from basic signal processing

Overheads

These Overheads and More at the **JOS Home Page**:

https://ccrma.stanford.edu/∼jos/Welcome.html#dsponline24

- Overview
- Deep Learning

Basic Idea

Architectures

Attention

History Samples

Example Deep Neural Network for F0 Estimation

Verma and Schafer, Interspeech 2016

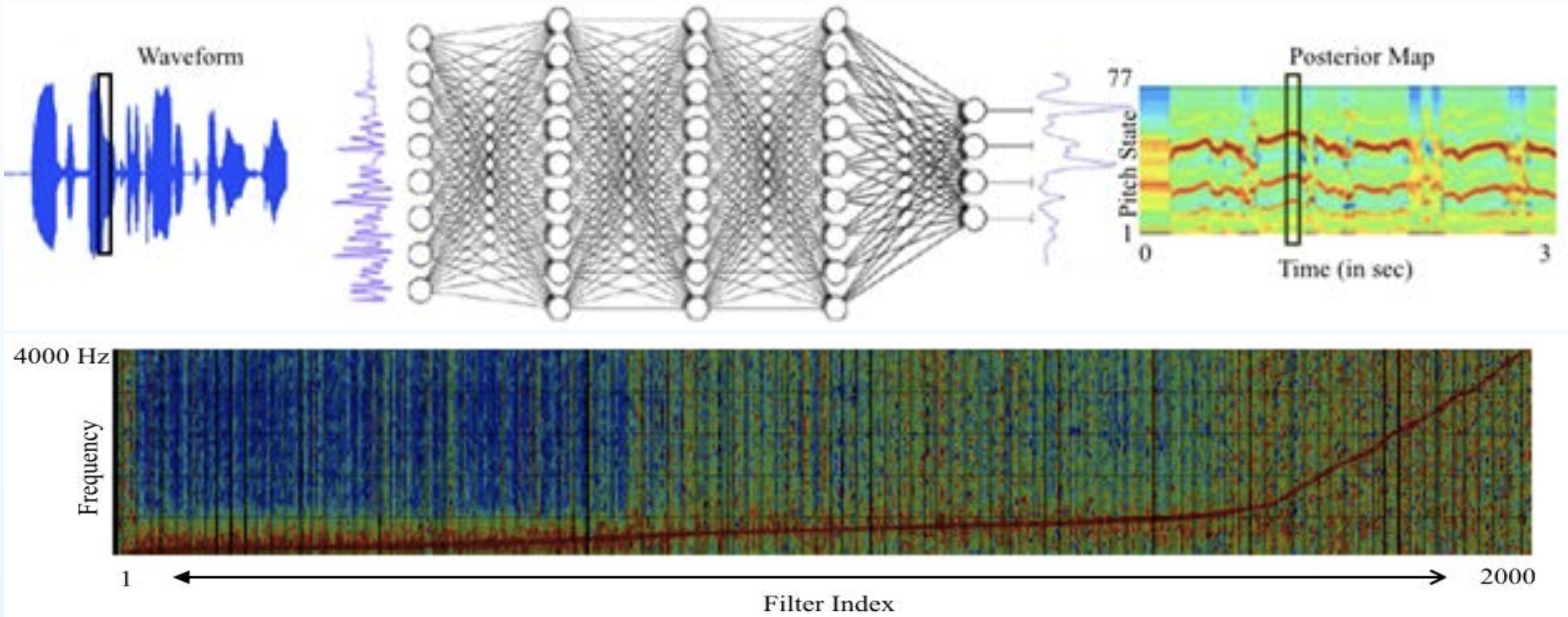
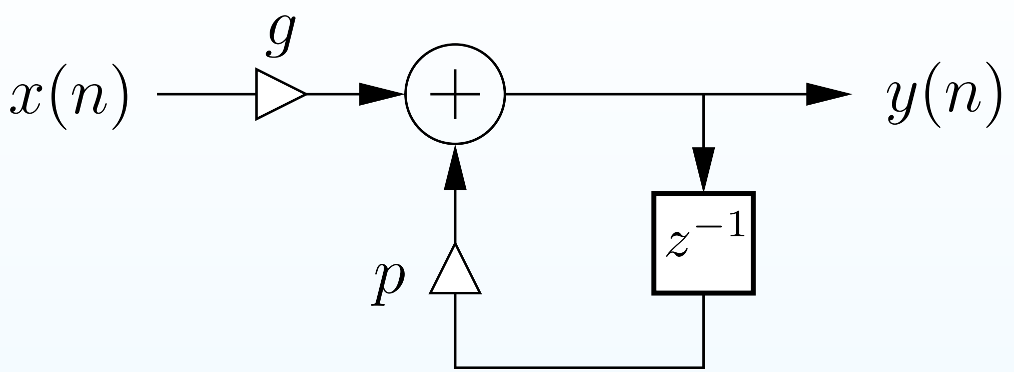


Figure 4: *Frequency response of the learned filters in the first layer sorted according to the highest peak for TIMIT. Red denotes high values whereas blue represents smaller value.*

- "Audio filter bank" *learned* in the first layer for the F0-estimation task
- Filter bands more dense in the F0 range
- Suggests: Replace first layer with a *pre-structured auditory filter bank* having a *differentiable and convex* parameterization, for *data* and *task adaptation* (see LEAF)

Motivating Problem: Associative Memory

Task: Make an *Associative Memory* using a *Vectorized One-Pole Filter*



One Pole at $z = p$

$$y(n) = g\,x(n) + p\,y(n-1),\, n = 0, 1, 2, \ldots \quad \text{[difference equation]}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{g}{1 - p\,z^{-1}} \quad \text{[transfer function]}$$

A Unit Delay (for Vectors) can be an Associative Memory:

- Generalize $x(n)$ to a *long vector* $\underline{x}(n) \in \mathbb{R}^N$ representing any "label"
- Set $\underline{g} = \underline{1}$ and $\underline{p} = \underline{1}$ to make $\underline{y}(n)$ a *sum of all input vectors* ("integrator")
- Choose the dimension N so large that *vectors in the sum are mostly orthogonal*
- Let $\underline{x}(n)$ be **embedding vectors** (*e.g.*, `word2vec`) so that *closeness = similarity*
- Retrieve similar vectors using a *matched inner product* $\underline{w}^T \underline{x} > b$, for some suitable threshold b (Hey! That's a simulated neuron! ("Perceptron" [1957]))

Vector Retrieval by Inner Product

Given the sum of vectors

$$\underline{y}(n) = \sum_{m=0}^{n} \underline{x}(m)$$

and a "query vector" $\underline{w} = \underline{x}(k)$,
find the query in the sum using an *inner product:*

$$\underline{w}^T \underline{y}(n) \;=\; \sum_{m=0}^{n} \underline{w}^T \underline{x}(m) \;\approx\; \underline{x}^T(k)\,\underline{x}(k) \;=\; \|\underline{x}(k)\|^2 \;>\; b(k)$$

or "found" if $\underline{w}^T \underline{y}(n) - b(k) > 0$, where $b(k)$ is the *detection threshold* for $\underline{x}(k)$

- This works because the spatial dimension is so large that $\underline{x}^T(j)\,\underline{x}(k) \approx \epsilon$ for $j \neq k$
- Retrieval threshold $b(k)$ depends on $\|\underline{x}(k)\|^2$
  $\Rightarrow$ **suggestion:** *reserve the radial dimension for similarity scoring*
- *I.e., only populate the* ***hypersphere*** in $\mathbb{R}^N$: $\|\,\underline{x}(k)\,\|^2 = 1$ (or N), $\forall k$
- We just invented `RMSNorm`, used extensively in neural networks (not `LayerNorm`)

Decaying Vector Retrieval by Inner Product

RNNs typically have a *forgetting factor* $p < 1$.

Given the sum of vectors

$$\underline{y}(n) = \sum_{m=0}^{n} p^{n-m} \underline{x}(m)$$

and a "query vector" $\underline{w} = \underline{x}(k)$,

the inner product now gives

$$\underline{w}^T \underline{y}(n) \;=\; \sum_{m=0}^{n} \underline{w}^T p^{n-m} \underline{x}(m) \;\approx\; p^{n-k} \underline{x}^T(k)\, \underline{x}(k) \;=\; p^{n-k} \;>\; b$$

where b is the detection threshold for $\underline{x}(k)$, independent of k if $\|\underline{x}(k)\| = 1$

- *Cannot retrieve* when $p^{n-k} < b$, setting an upper limit on n
- We need $p > b^{1/n}$ or $n_{\text{max}} \leq \log(b)/\log(p)$
- Lower $b \Rightarrow$ more memory, but also more false detections from "interference" ("other vectors in the sum")

Orthogonality in High Dimensions

Let $\mathbf{a} \in \mathbb{R}^N$ and $\mathbf{b} \in \mathbb{R}^N$ be two normally random, real, unit-norm vectors in N dimensions, with $\|\mathbf{a}\| = \|\mathbf{b}\| = 1$

The dot-product (inner product) of $\mathbf{a}^T = [a_1, a_2, \ldots, a_N]$ and $\mathbf{b}^T = [b_1, b_2, \ldots, b_N]$ is defined as

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T \mathbf{b} = \sum_{i=1}^{N} a_i b_i.$$

The squared dot product is

$$(\mathbf{a} \cdot \mathbf{b})^2 = \left(\sum_{i=1}^{N} a_i b_i\right)^2 = \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j b_i b_j.$$

Expected value (average):

$$E\left[(\mathbf{a} \cdot \mathbf{b})^2\right] = \sum_{i=1}^{N} \sum_{j=1}^{N} E[a_i a_j]\, E[b_i b_j] = \sum_{i=1}^{N} \frac{1}{N} \frac{1}{N} = \boxed{\frac{1}{N}}$$

- Overview
- Deep Learning

Basic Idea
- One Pole Filter
- Inner Product
- Orthogonality
- Language

Architectures

Attention

History Samples

Orthogonality in High Dimensions, Continued

We just showed the *expected squared dot product of two normally random unit vectors in* $\mathbb{R}^N$ *is* $1/N$, *i.e.*,

$$E\left[(\mathbf{a}\cdot\mathbf{b})^2\right] = \boxed{\frac{1}{N}}$$

since $E[a_i b_j] = 0$ for $i \neq j$, $E[a_i^2] = E[b_i]^2 = 1/N$, and $\mathbf{a}$ and $\mathbf{b}$ are independent.

Suggestions for Training:

- *Initialize biases (detection thresholds) near* $1/N$
- *Divide the sum of* M *vectors by* $\sqrt{M}$*:*
  - "power normalization"
  - "`RMSNorm`-preserving"
  - "Keep vector sums near the unit sphere"
- Apply `RMSNorm` when *training* the initial *vocabulary embedding* ("`word2sphere`")
- Set the *model dimension* just sufficient for the *layer width* at each level
- **Caveat:** We are only considering associative recall as *one mechanism* here. *Other mechanisms are definitely learned*, such as "attention sinks" and "induction heads", ...)

Orthogonality of Random Sums

Similarly,

$$
\begin{aligned}
E\left[\left(\underline{w}^T\underline{y}_n\right)^2\right] &= E\left[\left(\sum_{m=0}^{n}\underline{w}^T\underline{x}_m\right)^2\right] = \sum_{l=0}^{n}\sum_{m=0}^{n}E\left[\underline{w}^T\underline{x}_l\underline{x}_m^T\underline{w}\right] \\
&= \sum_{m=0}^{n}E\left[\underline{w}^T\underline{x}_m\underline{x}_m^T\underline{w}\right] = \sum_{m=0}^{n}E\left[\left(\underline{w}^T\underline{x}_m\right)^2\right] = \boxed{\frac{n}{N}}
\end{aligned}
$$

assuming $\underline{w} \notin \underline{y}$ and $\|\underline{w}\| = \|\underline{x}_m\| = 1$ for all m. Thus, *retrieval becomes unreliable when the number of summed vectors n nears the model dimension N.*

- N is of course the number of exactly orthogonal vectors possible in N dimensions
- If L vectors are typically in the sum, our Perceptron “bias” (detection threshold) should be higher than L/N
- **Suggestion:** *Keep the number of active vectors in memory well below N*

How Many Summed Vectors Needed for Language Parsing? (BERT style)

It is well known that *phone numbers* were limited to *7 digits* due to our *limited short-term memory* for *unrelated* objects. Can *language* be parsed using 7 vectors or less at each level? [Original Transformer paper had 8 attention heads and 6 layers (like neocortex)]

Layers (*e.g.*):

1. Base vocabulary = characters (26 for English)
2. Syllable in 7 chars or less (44 syllables in English; 107 in Int'l Phonetic Alphabet)
3. Word in 7 syllables or less
4. Noun + 6 or less modifying adjectives
5. Verb + up to 6 adverbs
6. Noun phrase
7. Direct or indirect object
8. Prepositional phrase
9. Subject, verb, [indirect object], object
10. Sentence
11. Paragraph
12. Section
13. Chapter
14. Book
15. Subject Area Hierarchy . . .

Different cortical areas (6 layers each) needed for this many levels.

Complex Sentence Diagram Examples:

`https://www.quora.com/In-regards-to-diagramming-sentences-which-one-is-the-most-difficult-youve-ever-come-across`

Pictorial Examples

- 6 dot-patterns on a die
- 7 LED segments for numbers
- 4 or fewer strokes to draw a letter

Hierarchy used to keep the number of components per concept *small*

Suggestions:

- *Ready signal* when symbol is recognized (whole letter, word, phrase, etc.)
- *Reset memory* after symbol recognition
- Memory can be *small!*

Maybe these signals are happening in the layers already?

Orthogonality of Exponentially Decaying Random Sums

RNNs typically have a *forgetting factor* $p < 1$, in which case we have, defining $\mu = n - m$ and $\lambda = n - l$:

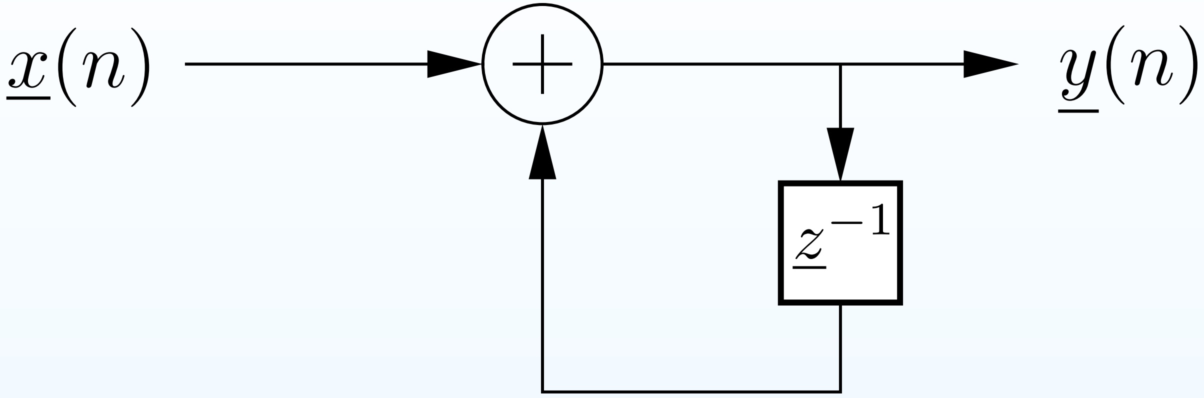
$$
\begin{aligned}
E\left[\left(\underline{w}^T \underline{y}_n\right)^2\right] &= E\left[\left(\sum_{m=0}^{n} \underline{w}^T p^{\mu} \underline{x}_m\right)^2\right] = \sum_{l=0}^{n}\sum_{m=0}^{n} E\left[\underline{w}^T p^{\lambda} \underline{x}_l p^{\mu} \underline{x}_m^T \underline{w}\right] \\
&= \sum_{m=0}^{n} p^{2\mu} E\left[\underline{w}^T \underline{x}_m \underline{x}_m^T \underline{w}\right] = \sum_{m=0}^{n} p^{2\mu} E\left[\left(\underline{w}^T \underline{x}_m\right)^2\right] \\
&= \boxed{\frac{1}{N}\frac{1-p^{2(n+1)}}{1-p^2}} \to \frac{1}{N}\frac{1}{1-p^2} \quad (\text{as } n \to \infty)
\end{aligned}
$$

- For $1/(1-p^2) < N$, keep $p < \sqrt{(N-1)/N}$
- For $1/(1-p^2) < N/2$, keep $p < \sqrt{(N-2)/N}$

and so on. *This gives us one way to calculate a maximum feedback coefficient p in RNNs*

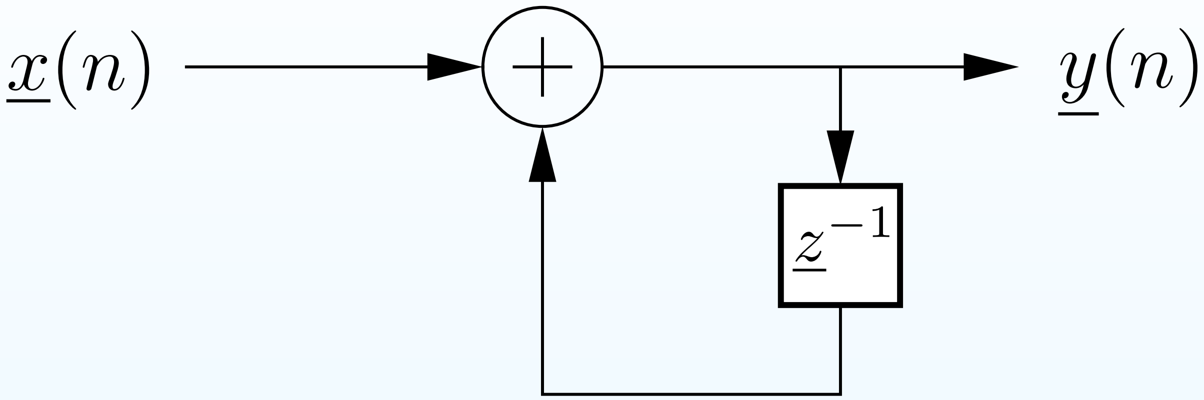
Architectures

Cumulative Vector Memory



MidJourney

Gated Vector Memory



Input Vector Summer

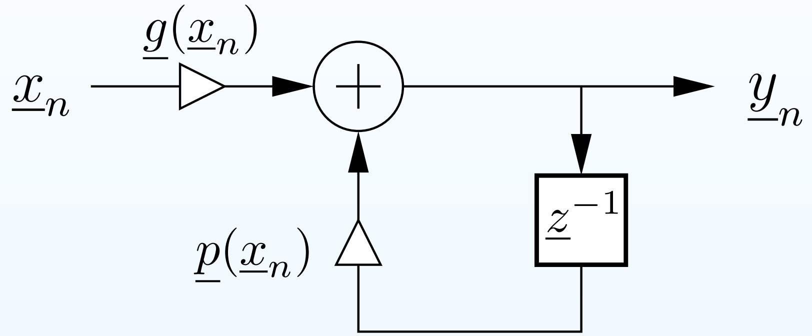
- **Problem:** Need a *memory reset*
- **Solution:** Set *feedback gain to zero* for one step to clear the memory

- **Problem:** Need an *input gate* to suppress unimportant inputs
- **Solution:** Set *input gain to zero* for unimportant inputs

- We just invented **gating**, used extensively in neural sequence models

Gated Recurrent Network

Idea: *Learn* the input and feedback gates as functions of input $\underline{x}_n$ based on many input-output examples $(\underline{x}_n, \underline{y}_n)$ ("training data"):



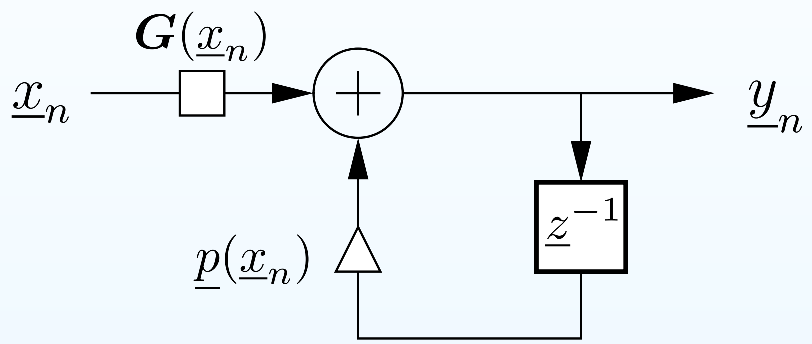
Vector Memory with Learned Input and Feedback Gates

Suggestions:

- Use learned, input-based, *activations* for gating (LSTM, GRU, Mamba)
- While activated, *optionally* set *memory duration* via $\underline{p}$ magnitude (SSMs, Mamba)
  - *Initialize* $\underline{p}$ for desired initial memory duration (exponential fade time)
  - Learn $\underline{p}(\underline{x}_n)$ as $\mathbf{I} \cdot e^{-\Delta} \approx \mathbf{I} - \mathbf{I}\Delta$, where $\Delta = \text{softPlus}(\text{parameter}(\underline{x}_n, \underline{y}_n))$ (guaranteed stable — no "exploding gradients") [Also multiply $\underline{g}(\underline{x}_n)$ by Δ]
  - Consider *separate meaning-driven activation* multiplying feedback: $\sigma(\mathbf{L}\underline{x})\underline{p}(\underline{x})$

Input Gating with Projection

Idea: Learn a *full matrix* for the input to provide *arbitrary projection* as well as *gating*



Vector Memory with Learned Input Projection and Gating

The added *linear transformation* can

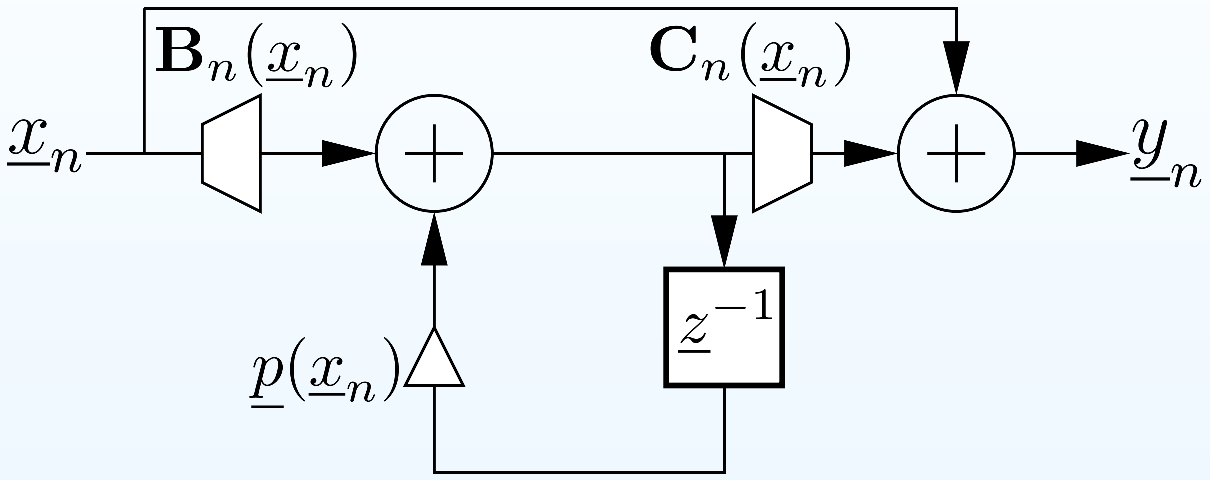
- further optimize the *input embedding* for the current *task* and *training data*,
- change the spatial layout to make room for things like temporal encoding,
- up-project to a higher internal model dimension (“state expansion” discussed later).

In *state-space models* such as S4 and H3,

- full *feedback matrices* $\mathbf{P}(\underline{x}_n)$ were investigated, but
- diagonal $\underline{p}(\underline{x}_n)$ were found to be sufficient (even a constant diagonal in Mamba-2).

Output Gating

Idea: Since we have input and feedback gates, why not an ***output gate* and bypass?**



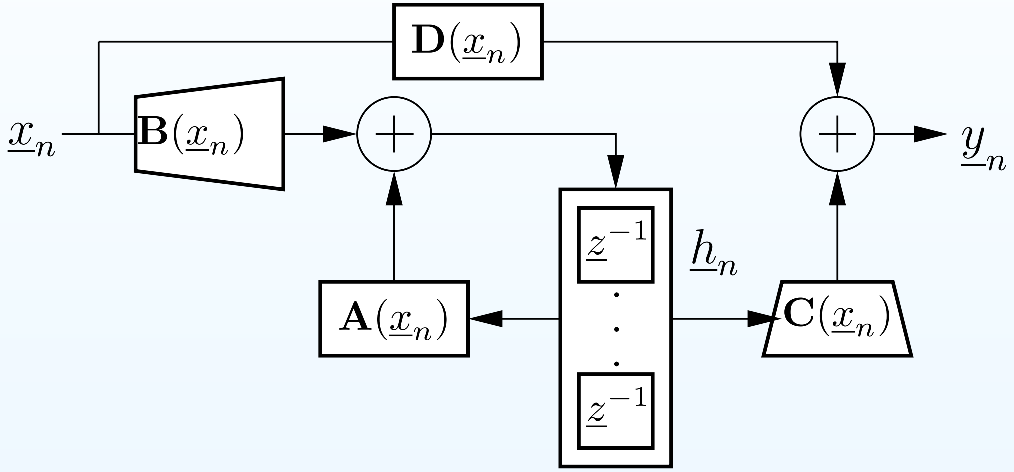
Gated RNN with **Skip Connection**

Output gating allows network to be “bypassed” when not helpful.

- **“Obvious” Suggestion:** The bypass path should be scaled for *power normalization*
- **Better yet:** Don’t scale the bypass and use `RMSNorm` at the input of the next layer (prevents a “bad layer” from isolating deeper layers from the input with garbage, and equalizes gradient backpropagation to all layers)

State Expansion

Idea: *Expand* vector-memory dimension to an integer multiple of the model dimension:



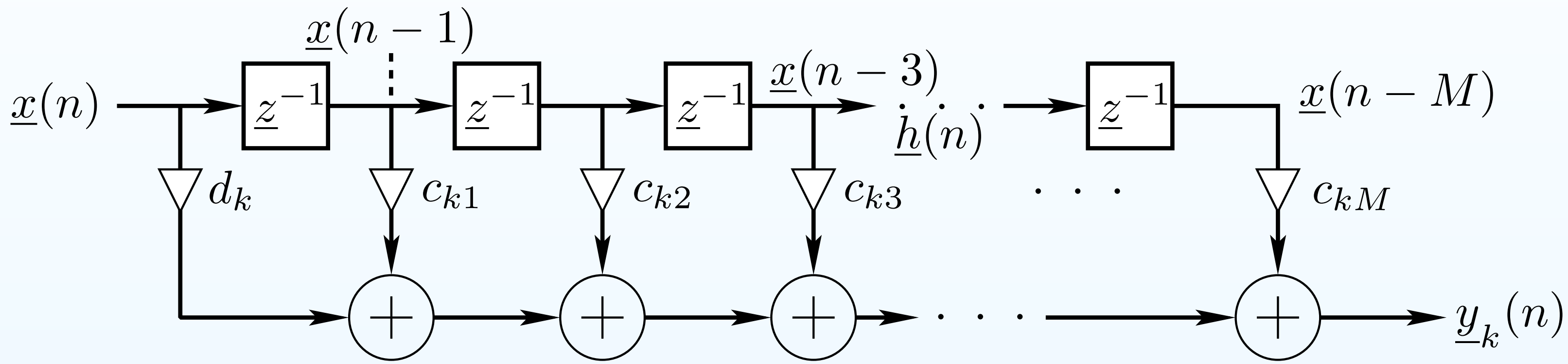
"Structured State-Space Models" (SSM) look like this (*e.g.*, Mamba)

- Increased storage capacity (more vectors can be summed and later retrieved)
- Feedback matrix $\mathbf{A}$ typically *diagonal* since 2022 (see "S4D")
  $\Rightarrow$ Parallel bank of vector one-poles (*"linearly" gated, state-expanded RNNs*)
- In Mamba-2, $\mathbf{A} = p\,\mathbf{I}$, *i.e.*, *shared memory duration* across expanded state
- Gating matrices in Mamba[-2] are simple linear input projections:
  $[\mathbf{B}(\underline{x}_n), \mathbf{C}(\underline{x}_n)] = \mathbf{L}\,\underline{x}_n$

Attention

Attention Layer

Idea: Also use *FIR Filtering* (SSM State Expansion Factor M, $\mathbf{A}$ subdiagonal):



Separately learnable FIR coefficient matrices $d_k[\underline{x}(n)], c_j[\underline{x}(n-j), k]$, depending on:

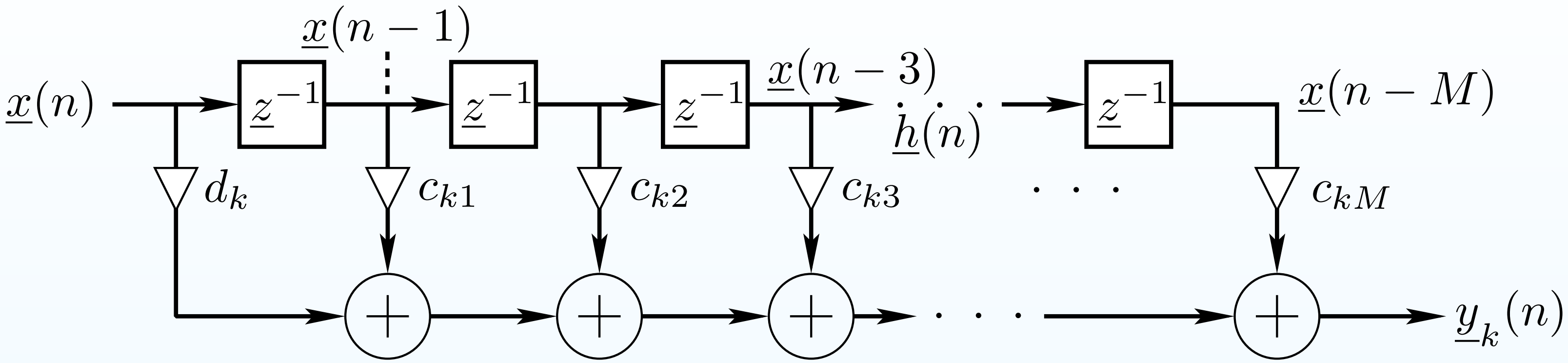
1. input *position* j in the input sequence ("context buffer" or "expanded state" + [W]RoPE)
2. input *vector* $\underline{x}(n-j), \;\; j = 0, 1, 2, \ldots, M$
3. *output-position* k being computed, $\; k = 0, 1, 2, \ldots, M$ ($M + 1$ outputs)

Idea: Add *relevance gating* suppressing unimportant inputs to each output ("attention")

Idea: Create *new embedding vectors* as *sums* of relevant input vectors ("attention")

Idea: Measure relevance using an *inner product* between the output and input positions ("dot-product attention")

Dot-Product Attention



Relevance Gating

Let $\underline{x}_k$ denote $\underline{x}(n-k)$

The contribution from input $\underline{x}_j$ to the nonlinear FIR sum for output $\underline{y}_k$ can be calculated as

$$c_{kj}\underline{x}_j = \left[\left(\sum_{m\in\mathcal{R}(\underline{x}_k)} \underline{x}_m\right)^T \underline{x}_j\right] \underline{x}_j$$

or more generally $\boxed{c_{kj} = \underline{q}_k^T \underline{x}_j}$, where

$\underline{q}_k$ is called the *query* vector for position k in the output sequence

The query $\underline{q}_k$ can be for example a *sum of vectors allowed in the attention sum*:

$\underline{q}_k = \underline{x}_k + \underline{x}_{m_1} + \cdots + \underline{x}_{m_k}$

$\Rightarrow (\underline{q}_k^T \underline{x}_j)\underline{x}_j \approx \underline{x}_j$, if $\underline{x}_j$ is similar to *any vector* in the query sum.

Multi-Head Attention

Idea: To support multiple meaning possibilities, *partition the model space* into parallel independent *attention calculations* (“multi-head attention”)

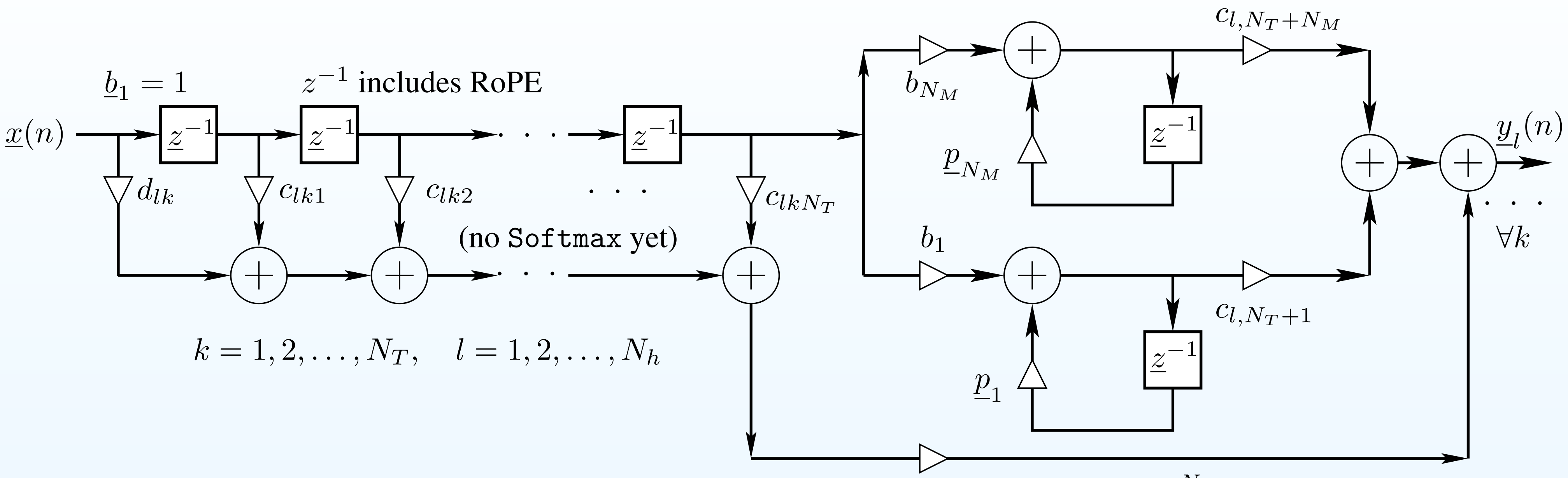
- Each *attention head* can form an independent input interpretation
- Useful for *ambiguous* sequences, especially in the lower layers
- Also introduced in the Transformer paper (2017)

Now we need *down-projections* of the relevance-calculation components
$\Rightarrow$ relevance of input j to output k in attention-head l becomes proportional to

$$c_{kj}\underline{x}_j = (\underline{q}_k^T \underline{x}_j)\underline{x}_j \;\longrightarrow\; c_{lkj}\underline{x}_{lj} = \left[\underline{q}_{lk}^T(\underline{x}_k)\, k_{lj}(\underline{x}_j)\right] v_{lj}(\underline{x}_j)$$

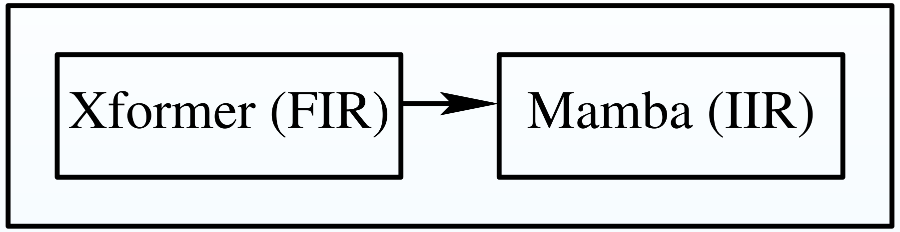
where $\underline{q}_{lk}$ (“query”), k_{lj} (“key”), and v_{lj} (“value”) vectors are learned *down-projections* of the input $\underline{x}_j$ for each attention-head l and for all sequence indices j and k in the context buffer (“Transformer”)
Other useful generalizations can be imagined for these learned (Q,K,V) vectors, such as grouping grammatical functions, creating new model-space regions, etc.

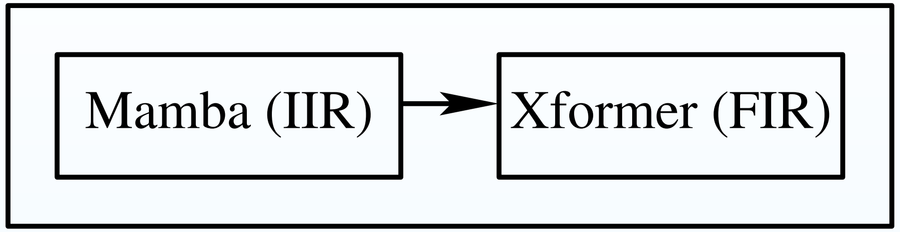
Transformer followed by GRNN with 2x State Expansion (like Mamba)



$$\text{Consider } \frac{1}{\sqrt{\sum_{j=1}^{N_T} (Q_k^T K_j)^2}} \Rightarrow \sum_{j=1}^{N_T} c_{lkj}^2 = 1$$

"Direct Form I" or "Direct Form II"?





XMamba ("DF-I") versus MambaX ("DF-II")

- Perfect short-term memory
- Fuzzy, fading, long-term memory
- Like Infini-Attention

- Both memories see latest input
- IIR part less efficiently used

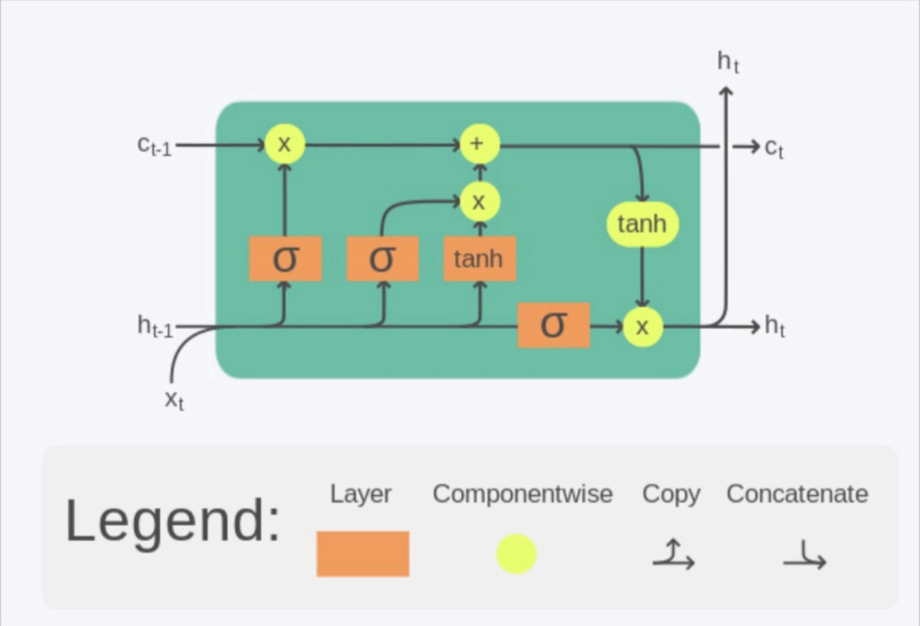
Direct Form I looks preferable because it separates short and longer-term memory functions.

Thanks for your Attention!

Sequence Modeling Snapshots

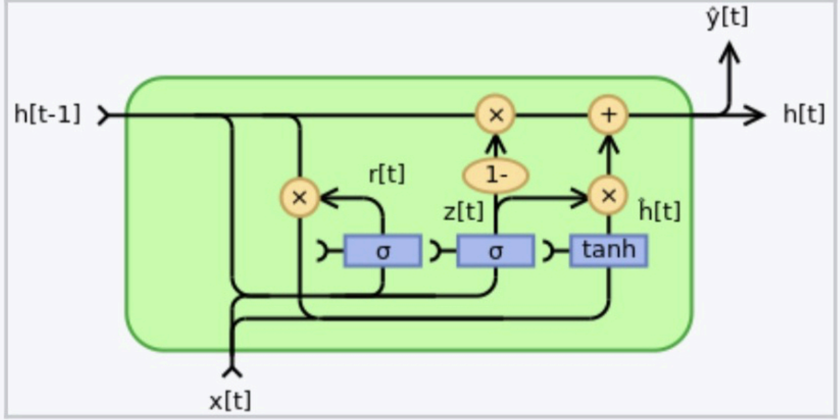
LSTM and GRU

1997: LSTM



The Long Short-Term Memory (LSTM) cell can process data sequentially and keep its hidden state through time.

2014: GRU



Gated Recurrent Unit, fully gated version

Structured State Space and Mamba

2023: Mamba (S6)

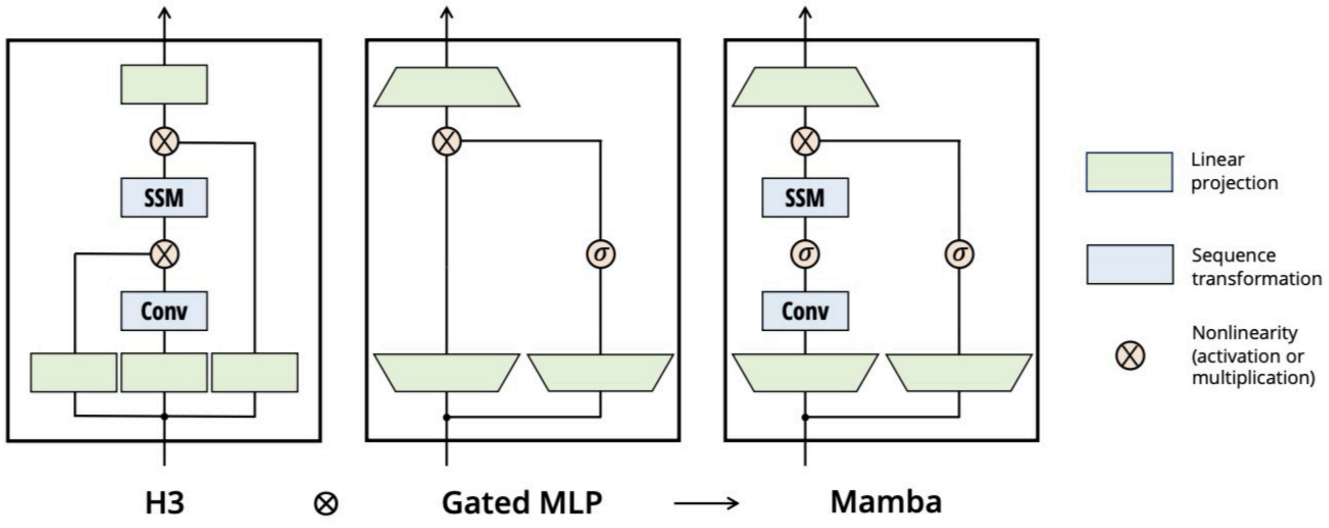


Figure 2: (**Architecture.**) Our simplified block design combines the H3 block, which is the basis of most SSM architectures, with the ubiquitous MLP block of modern neural networks. Instead of interleaving these two blocks, we simply repeat the Mamba block homogenously. Compared to the H3 block, Mamba replaces the first multiplicative gate with an activation function. Compared to the MLP block, Mamba adds an SSM to the main branch. For σ we use the SiLU / Swish activation (Hendrycks & Gimpel, 2016; Ramachandran et al., 2017).

Hawk and Griffin

Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models

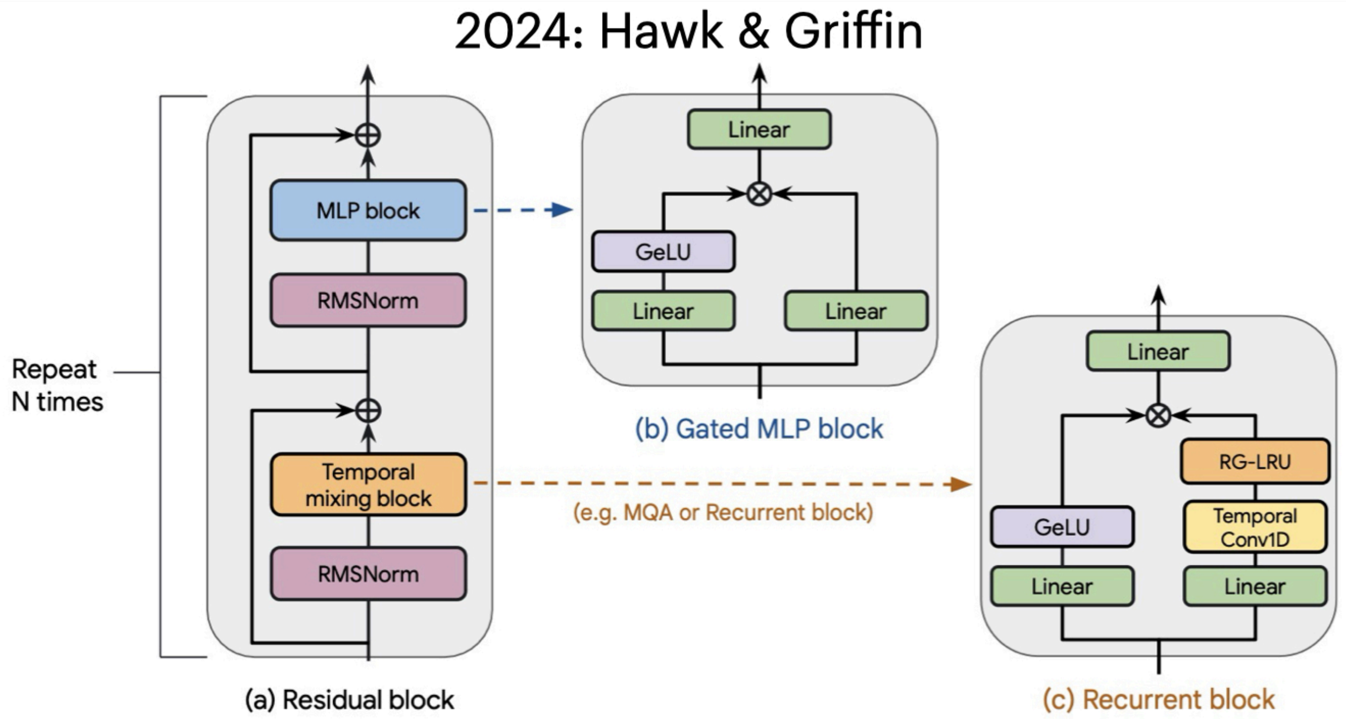


Figure 2 | a) The main backbone of our mode architecture is the residual block, which is stacked N times. b) The gated MLP block that we use. c) The recurrent block that we propose as an alternative to Multi Query Attention (MQA). It uses our proposed RG-LRU layer, defined in Section 2.4.

Gated "Linear" RNNs with State Expansion

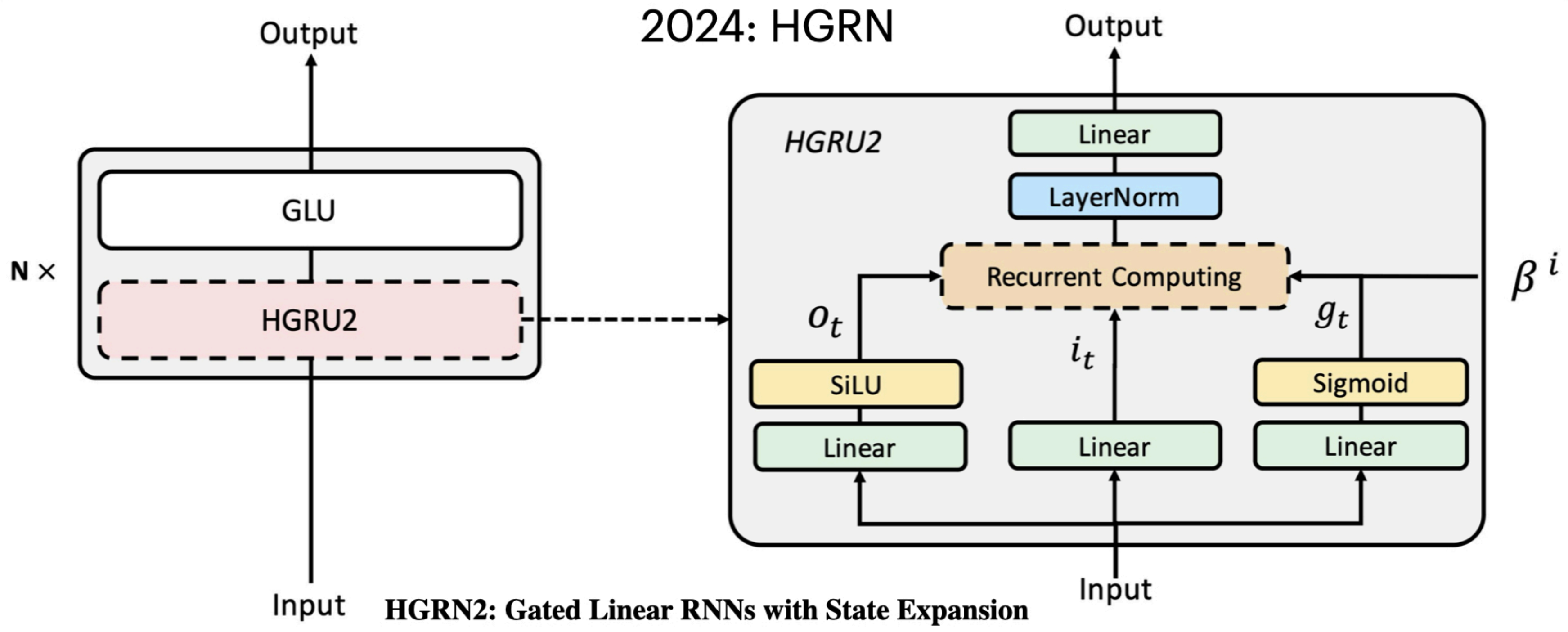


Figure 1: **The neural architecture of HGRN2.** Each HGRN2 layer includes a token mixer layer HGRU2 and a channel mixerlayer GLU. HGRU2 employs recurrent computation through Eq. 3, where $\mathbf{i}_t$ is the input vector, $\mathbf{g}_t$ is the forget gate (not lower bounded), β^i is the lower bound of the forget gate value, $\mathbf{o}_t$ is the output gate for layer i.

RWKV, Eagle, Finch

2024: Eagle-Finch RWKV

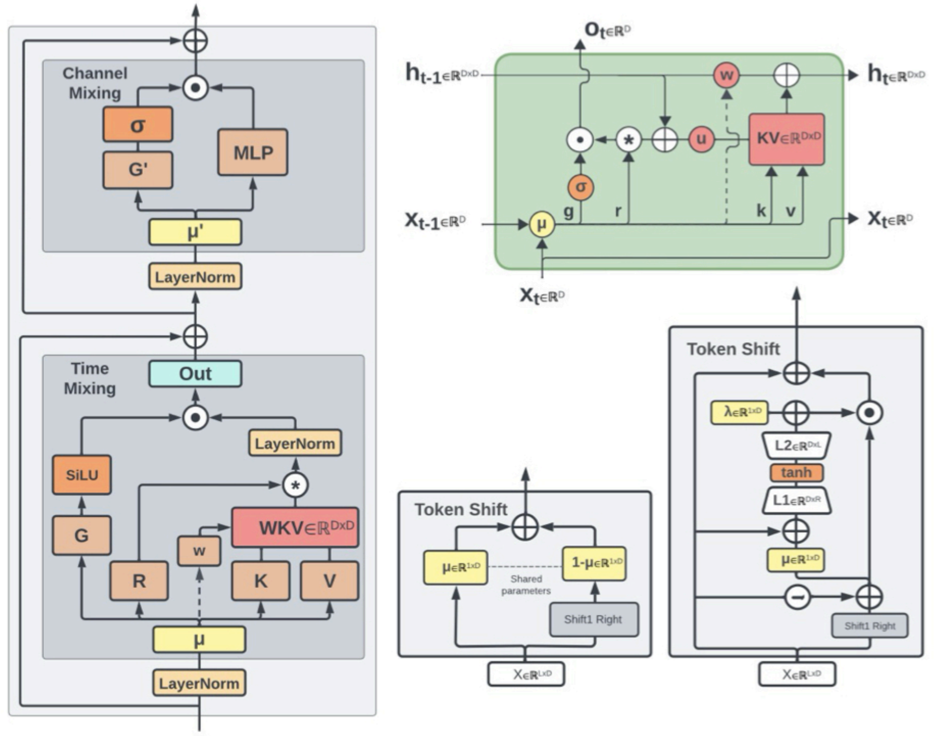


Figure 1: RWKV architecture overview. **Left:** time-mixing and channel-mixing blocks; **top-right:** RWKV time-mixing block as RNN cell; **center-bottom:** token-shift module in FeedForward module and Eagle time-mixing; **bottom-right:** token-shift module in Finch time-mixing. All shape annotations assume a single head for simplicity. Dashed arrows (left, top-right) indicate a connection in Finch, but not in Eagle.

Jamba, Zamba, & Samba Hybrid Architectures (Mamba then Attention)