Lecture 8

Architecture 1: Signal propagation in time

CS312

Tatsu H

Reminders

A4 (Transformers vs RNNs) due Thu 10/29, **1pm**

Quiz Thu 10/29, in class

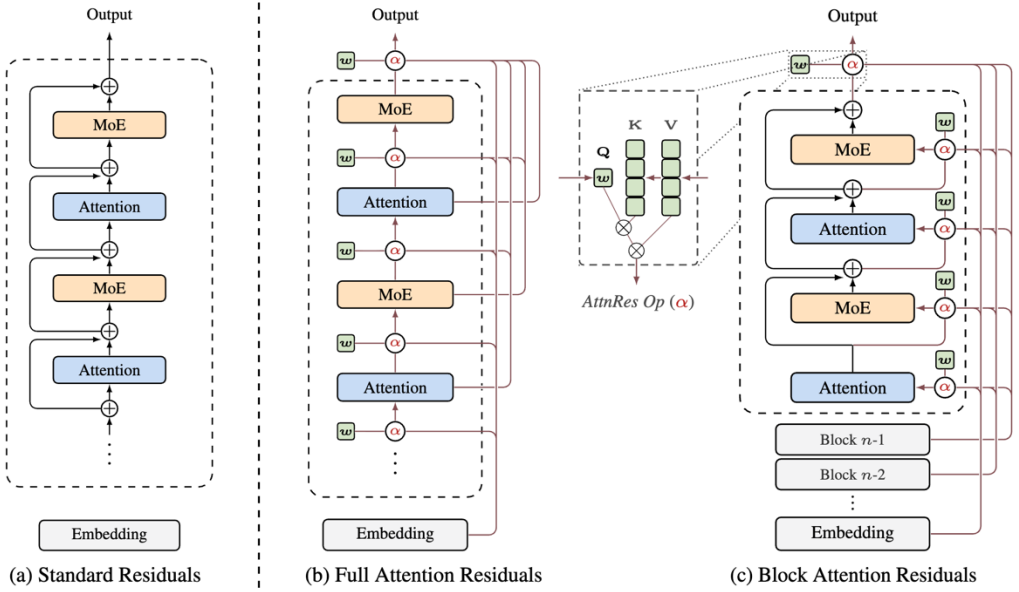
From last lecture: attention residuals

ATTENTION RESIDUALS

TECHNICAL REPORT OF ATTENTION RESIDUALS

Kimi Team
https://github.com/MoonshotAI/Attention-Residuals

Allow for weighted averages from the distant past



$$\boldsymbol{h}_l = \alpha_{0\to l} \cdot \boldsymbol{h}_1 + \sum_{i=1}^{l-1} \alpha_{i\to l} \cdot f_i(\boldsymbol{h}_i)$$

$$\alpha_{i\to l} = \frac{\phi(\boldsymbol{q}_l, \boldsymbol{k}_i)}{\sum_{j=0}^{l-1} \phi(\boldsymbol{q}_l, \boldsymbol{k}_j)}$$

$$\boldsymbol{q}_l = \boldsymbol{w}_l, \qquad \boldsymbol{k}_i = \boldsymbol{v}_i = \begin{cases} \boldsymbol{h}_1 & i = 0 \\ f_i(\boldsymbol{h}_i) & 1 \le i \le l-1 \end{cases}$$

Query is a learned vector per layer. Key and values are the ‘residual output’

Details with attention residuals

No longer identity init

Suggested initialization is *uniform* over previous layers. Very much not identity

Could enable ‘dead layers’

Softmax could concentrate on a few layers, or skip layers

In practice, seems to work well

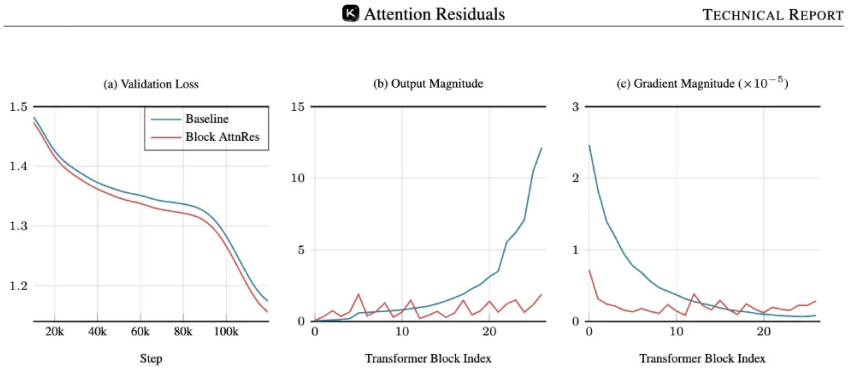


Table 4: Ablation on key components of AttnRes (16-layer model).

Variant		Loss
Baseline (PreNorm)		1.766
DenseFormer [36]		1.767
mHC [59]		1.747
AttnRes	Full	1.737
	w/ input-dependent query	1.731
	w/ input-independent mixing	1.749
	w/ sigmoid	1.741
	w/o RMSNorm	1.743
	SWA ($W = 1 + 8$)	1.764
	Block ($S = 4$)	1.746
	w/ multihead ($H = 16$)	1.752
	w/o RMSNorm	1.750

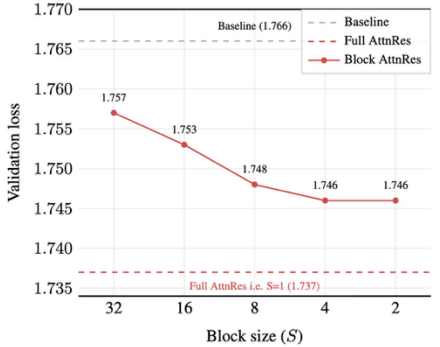


Figure 6: Effect of block size on validation loss (16-layer model).

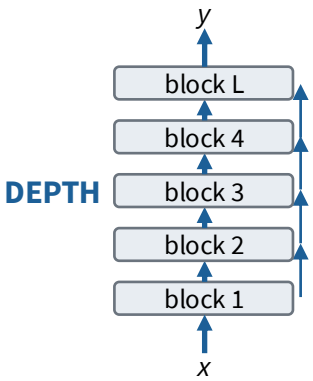
Today's lecture: signal propagation (and mixing) in time

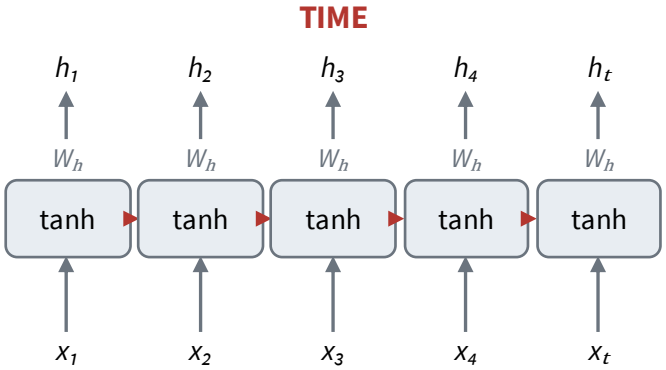
Last lecture: Signal propagation in *depth* . What were the key concerns?

- Getting extremely deep networks to train (1000+ layers)
- Understanding the fundamentals (Jacobian, variance tracking)

This lecture: Signal propagation in *time* . Isn't this the same thing?

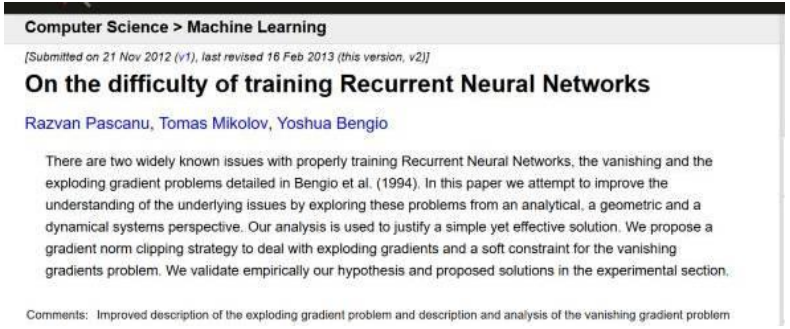
- Very much not! Surprisingly the solutions are very different and diverse
- Why? One reason: we have *position dependent inputs* and *weight sharing*





RNNs were hard to train well

Similar development to resnets – RNNs (and LSTMs) were difficult to train well



Computer Science > Machine Learning

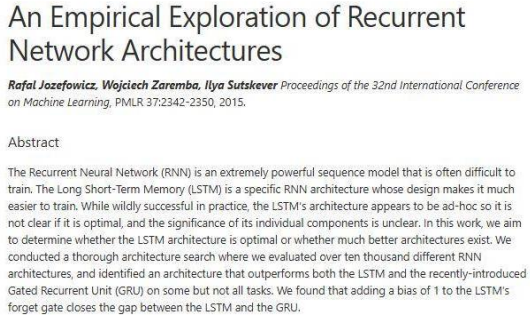
[Submitted on 21 Nov 2012 (v1), last revised 16 Feb 2013 (this version, v2)]

On the difficulty of training Recurrent Neural Networks

Razvan Pascanu, Tomas Mikolov, Yoshua Bengio

There are two widely known issues with properly training Recurrent Neural Networks, the vanishing and the exploding gradient problems detailed in Bengio et al. (1994). In this paper we attempt to improve the understanding of the underlying issues by exploring these problems from an analytical, a geometric and a dynamical systems perspective. Our analysis is used to justify a simple yet effective solution. We propose a gradient norm clipping strategy to deal with exploding gradients and a soft constraint for the vanishing gradients problem. We validate empirically our hypothesis and proposed solutions in the experimental section.

Comments: Improved description of the exploding gradient problem and description and analysis of the vanishing gradient problem

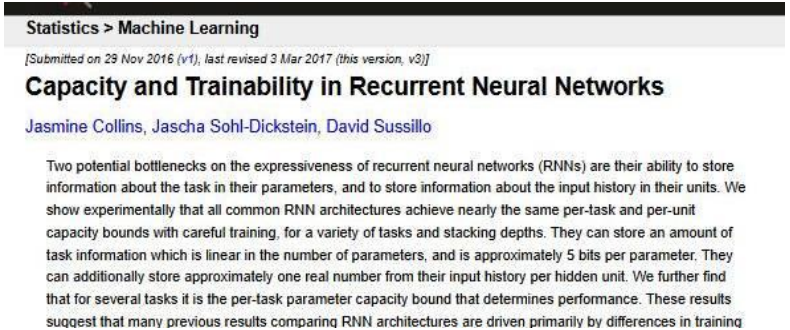


An Empirical Exploration of Recurrent Network Architectures

Rafal Jozefowicz, Wojciech Zaremba, Ilya Sutskever Proceedings of the 32nd International Conference on Machine Learning, PMLR 37:2342-2350, 2015.

Abstract

The Recurrent Neural Network (RNN) is an extremely powerful sequence model that is often difficult to train. The Long Short-Term Memory (LSTM) is a specific RNN architecture whose design makes it much easier to train. While wildly successful in practice, the LSTM's architecture appears to be ad-hoc so it is not clear if it is optimal, and the significance of its individual components is unclear. In this work, we aim to determine whether the LSTM architecture is optimal or whether much better architectures exist. We conducted a thorough architecture search where we evaluated over ten thousand different RNN architectures, and identified an architecture that outperforms both the LSTM and the recently-introduced Gated Recurrent Unit (GRU) on some but not all tasks. We found that adding a bias of 1 to the LSTM's forget gate closes the gap between the LSTM and the GRU.



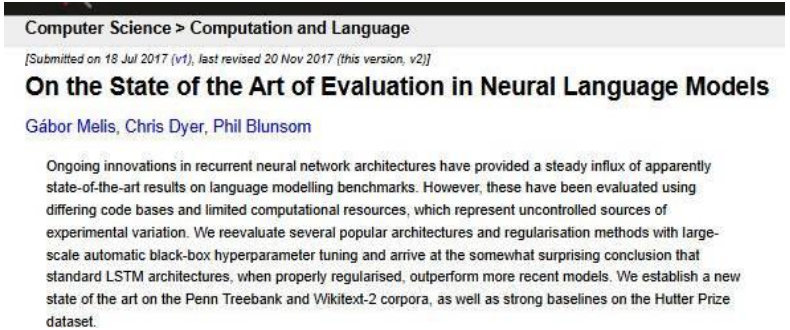
Statistics > Machine Learning

[Submitted on 29 Nov 2016 (v1), last revised 3 Mar 2017 (this version, v3)]

Capacity and Trainability in Recurrent Neural Networks

Jasmine Collins, Jascha Sohl-Dickstein, David Sussillo

Two potential bottlenecks on the expressiveness of recurrent neural networks (RNNs) are their ability to store information about the task in their parameters, and to store information about the input history in their units. We show experimentally that all common RNN architectures achieve nearly the same per-task and per-unit capacity bounds with careful training, for a variety of tasks and stacking depths. They can store an amount of task information which is linear in the number of parameters, and is approximately 5 bits per parameter. They can additionally store approximately one real number from their input history per hidden unit. We further find that for several tasks it is the per-task parameter capacity bound that determines performance. These results suggest that many previous results comparing RNN architectures are driven primarily by differences in training



Computer Science > Computation and Language

[Submitted on 18 Jul 2017 (v1), last revised 20 Nov 2017 (this version, v2)]

On the State of the Art of Evaluation in Neural Language Models

Gábor Melis, Chris Dyer, Phil Blunsom

Ongoing innovations in recurrent neural network architectures have provided a steady influx of apparently state-of-the-art results on language modelling benchmarks. However, these have been evaluated using differing code bases and limited computational resources, which represent uncontrolled sources of experimental variation. We reevaluate several popular architectures and regularisation methods with large-scale automatic black-box hyperparameter tuning and arrive at the somewhat surprising conclusion that standard LSTM architectures, when properly regularised, outperform more recent models. We establish a new state of the art on the Penn Treebank and Wikitext-2 corpora, as well as strong baselines on the Hutter Prize dataset.

Pascanu et al. 2013; Jozefowicz et al. 2015; Collins et al. 2017; Melis et al. 2017

Our overall view: generalizing the end-to-end Jacobian

From last lecture: end to end jacobian

$$J_{ij}^{N_l,1}(x^{N_l}) \equiv \left.\frac{\partial x_i^{N_l}}{\partial x_j^1}\right|_{x^{N_l}}$$

This represents how gradients at the last layer propagate to the earlier (or first) layer.

The spectrum of the Jacobian tell us how the matrix shrinks or explodes gradients across layers

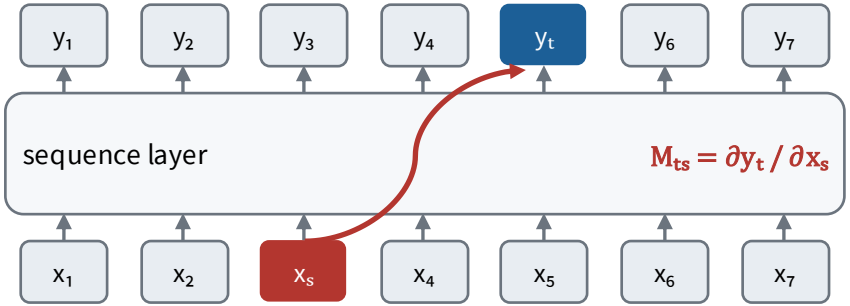
This lecture: a ‘more general’ view of the end to end Jacobian

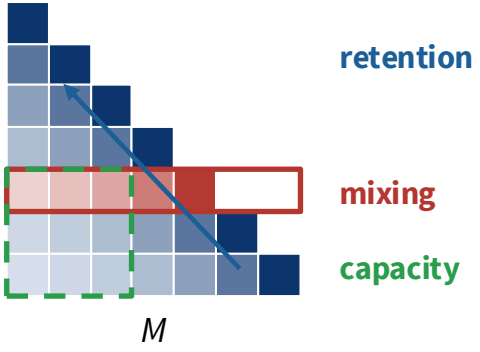
A generalized end-to-end Jacobian

At each output (t), we now have many inputs from the 'causal' structure (t' < t)

- **Same as last time:** how well does information travel between any *pair* of inputs (**Retention**)
- **New stuff:** interactions between different inputs (**Mixing**) and how many inputs propagate (**Capacity**)

$$M_{ts} = \frac{\partial y_t}{\partial x_s} \in \mathbb{R}^{d \times d}, \qquad M = \begin{pmatrix} M_{11} & 0 & \cdots & 0 \\ M_{21} & M_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ M_{T1} & M_{T2} & \cdots & M_{TT} \end{pmatrix}$$





Core questions about retention, mixing, capacity

Retention: What types of architectures allow for long-range signal propagation?

Mixing: How do we ensure that we can store and retrieve the ‘right’ piece of information

Capacity: What are the best ways of storing and propagation many bits of information?

And… How do we manage the tradeoff among these goals??

Lecture today

Part 1

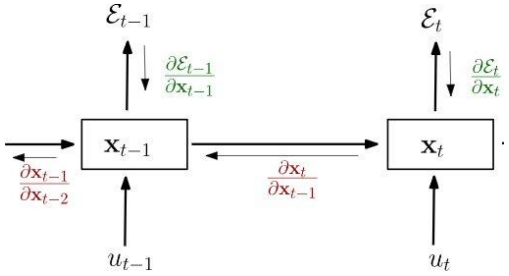
Retention: RNNs to LSTMs

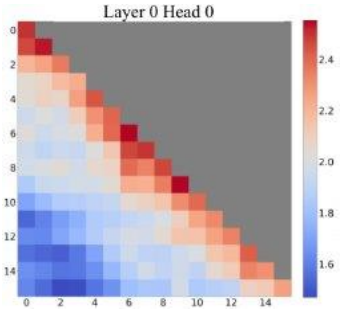
Part 2

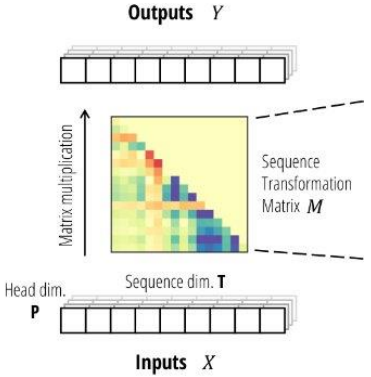
Mixing: attention

Part 3

Capacity: SSMs vs LSTMs



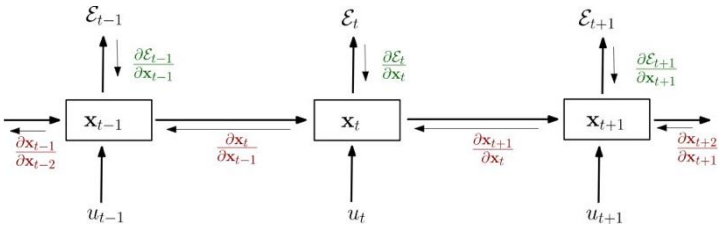




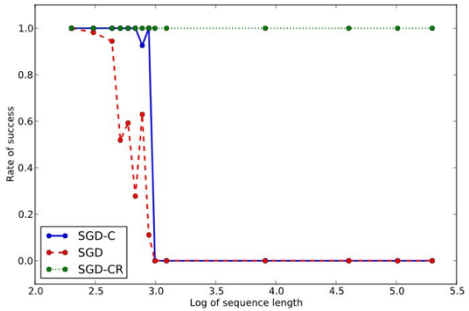
The RNN is a very deep network across time

RNNs look like a very deep network with: new inputs (x) and tied weights

$$h_t = \tanh(W_x x_t + W_h h_{t-1}), \qquad \frac{\partial h_t}{\partial h_s} = \prod_{r=s+1}^{t} \mathrm{diag}(\tanh'(a_r)) W_h$$



- The gradient across time is a product of per-step Jacobians. These are unstable (even more so with tied weights!)
- **In practice,** RNNs are hard to train, especially for 'long context' tasks



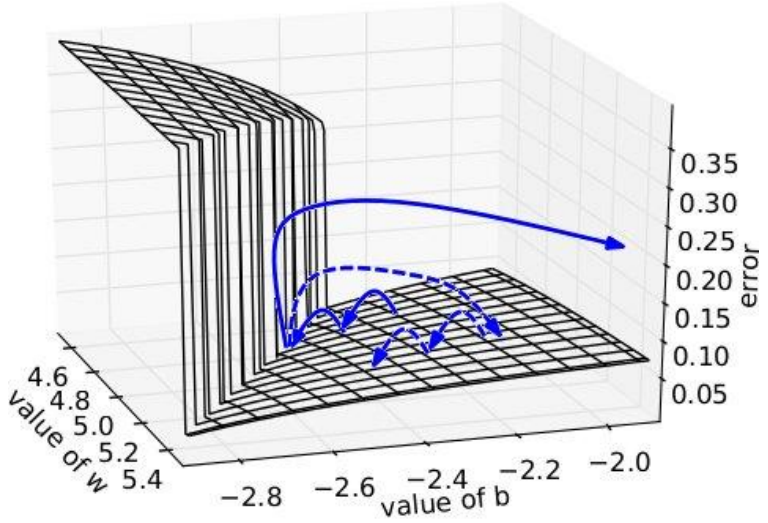
Pascanu, Mikolov & Bengio 2013

Similar failure modes as depth

Exploding and vanishing gradients – exponential divergence from unit singular value

$$\| \frac{\partial h_t}{\partial h_k} \| \leq (\gamma \sigma_{\max}(W_h))^{t-k}$$

- The failure modes of RNNs are very similar to very deep networks
- Observation similar to last week that there was a very narrow 'window of stability'
- **RNN trick:** clip gradients to partially control the upper half (exploding). But your gradients are still degenerate + can vanish



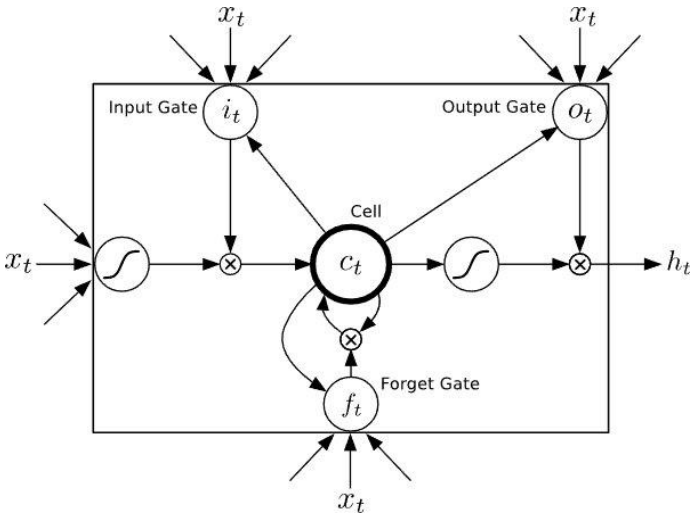
Pascanu, Mikolov & Bengio 2013

Why not have a ‘residual’?

Hochreiter and Schmidthuber 1997: a more direct, cleaner signal path across time

$$c_t = c_{t-1} + i_t \odot g_t, \qquad h_t = o_t \odot \tanh(c_t)$$

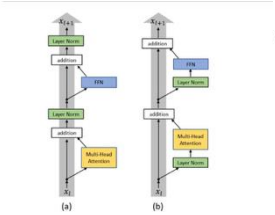
- **Constant error carousel** (c): A residual connection where the ‘error’ (cotangent) flows back across time
- **Input gated writes** (i): Control how much to change the state
- **Per step outputs** (h) are also gated
- Learns models with 1,000+ steps, like a ResNet

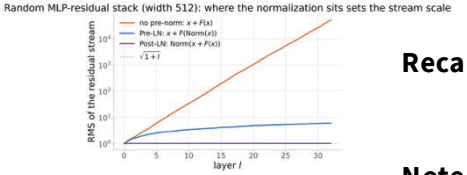


Hochreiter & Schmidhuber 1997; figure from Graves 2013

Variance growth and branch ratios

Challenge from last lecture: it's hard to balance the variance of the skip and residual





Recall: post-norm scales the residual, leads to later-layer bias. pre-norm the opposite

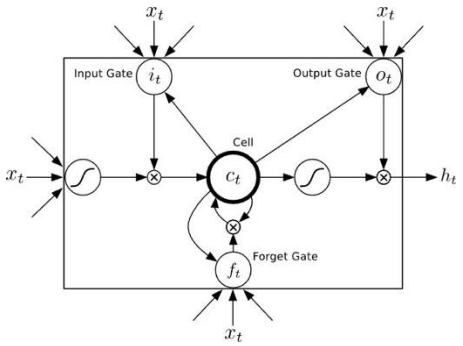
Note: layer norm was originally invented to handle the input scales in a RNN!

Recurrent network solution: have a 'forget gate' that can downscale the 'residual'

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

Why do this instead of what we did before?
Depth case: dead units are wasted compute
Time case: dead units are inputs we drop

It's more important to 'use all layers'!



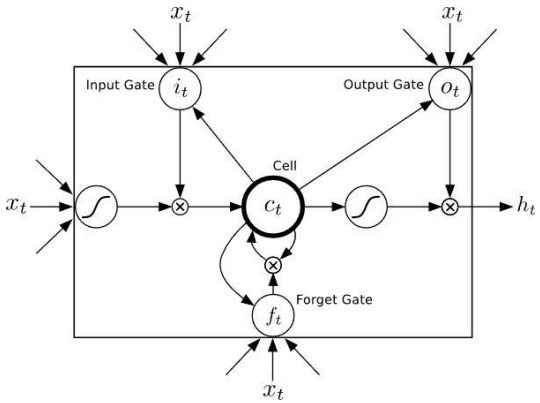
Gers, Schmidhuber & Cummins 2000

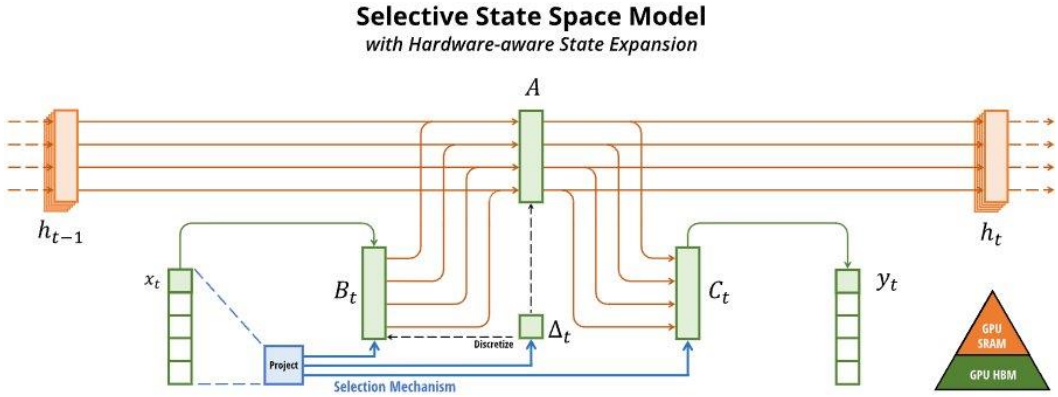
These ideas are the foundation of today's recurrent models

Signal propagation principles are similar across architectures

- LSTM has gated recurrence with gates *dependent on the previous outputs* (nonlinear)
- More recent SSMs have linear temporal dependence (and nonlinear input gating).

LSTM: $c_t = f_t \odot c_{t-1} + i_t \odot g_t$ Mamba: $h_t = e^{-\Delta_t A} \odot h_{t-1} + \Delta_t B x_t$





Graves 2013; Gu & Dao 2023

Side remark: can the same tricks remove gating?

The tricks that worked well in depth don't work particularly well for sequence modeling

	Idea	Toy tasks	Gains for LMs
IRNN / unitary RNN	orthogonal or unitary W_h	yes, T = 1,000+	no
Critical init	Edge of chaos init	yes, T = 784	no: PTB 89.9 vs 83.8
S4 / LRU	diagonal timescales	yes, incl. Path-X	no (until Mamba)

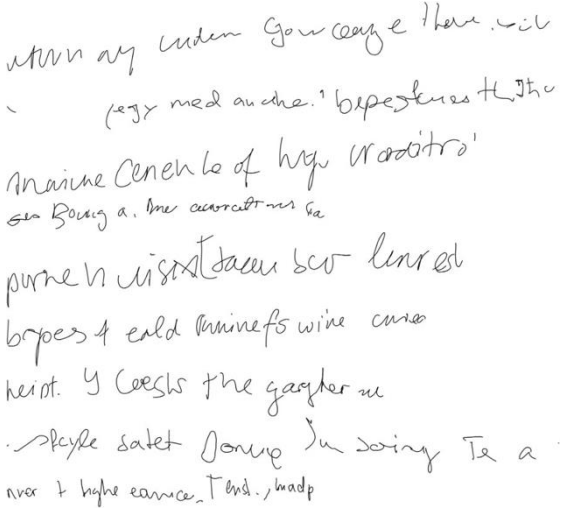
In sequence modeling we need more than just initialization-time stability across time.

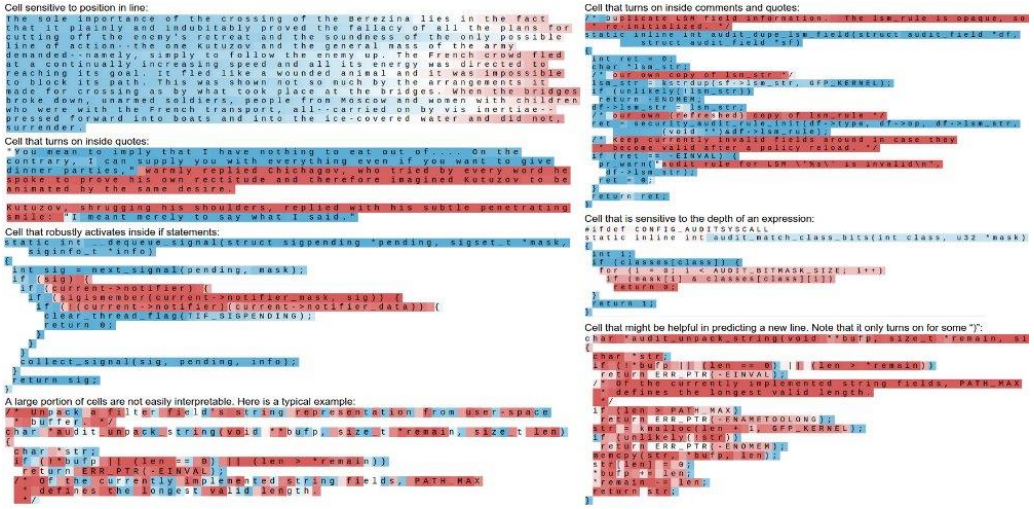
Le et al. 2015; Arjovsky et al. 2016; Chen et al. 2018; Gu et al. 2021; Orvieto et al. 2023

Advances in sequence models with LSTMs

~2015: Many interesting successes with LSTMs

- **Left**: Generating handwriting (real valued, sequential output)
- **Right**: Generating text, with LSTM cells tracking interpretable features





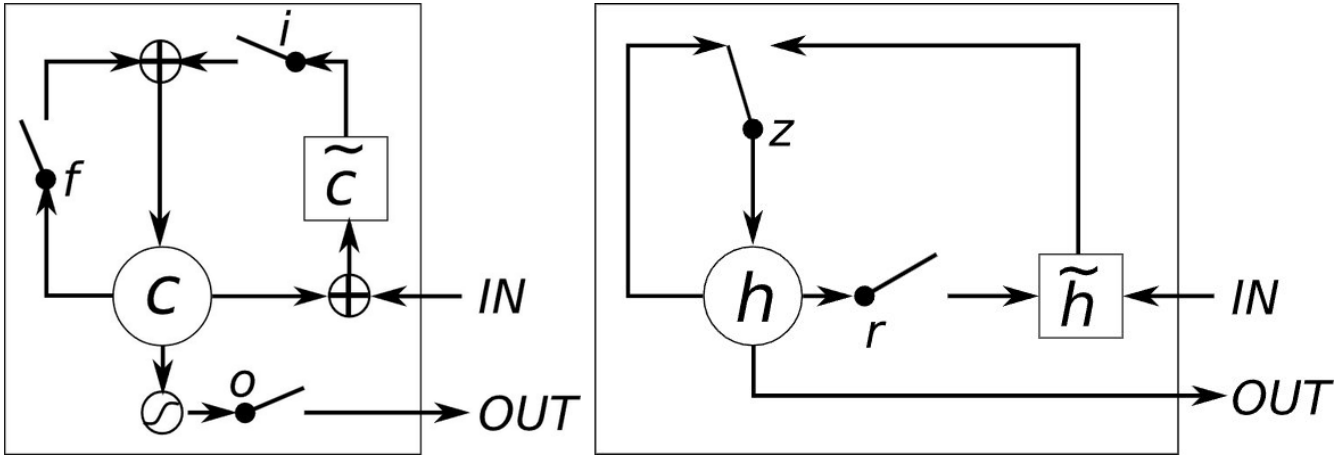
Graves 2013; Karpathy, Johnson & Fei-Fei 2015

Simplifying the LSTM (GRU)

Merge the forget and input gates, merge ‘c’ and ‘h’ (cell and output)

- Can merge the two if the goal is to forget only when necessary for new information
- In theory, only need one type of ‘recurrent state’ to carry information.

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t, \qquad \tilde{h}_t = \tanh(W x_t + U(r_t \odot h_{t-1}))$$

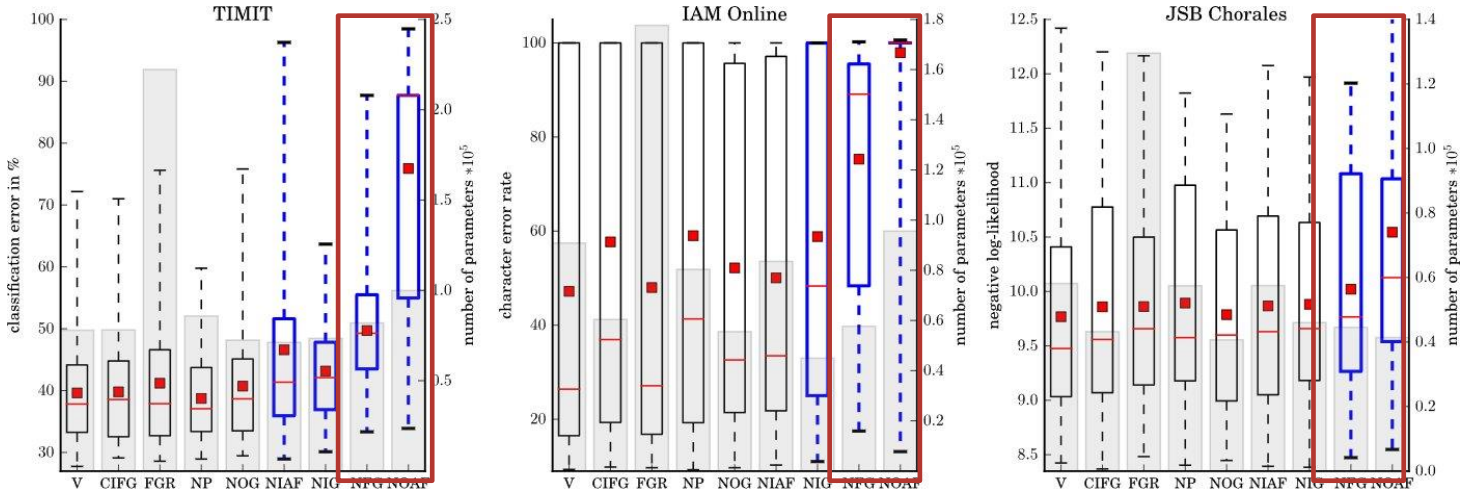


Cho et al. 2014; Chung et al. 2014

So, what actually matters?

Large sweeps (2015): only the forget gate and the output tanh are essential

- **Greff et al.**: 8 variants × 3 tasks × 200 runs
  - CIFG (coupling input + forget), NP (no peephole) have no impact
  - NFG (no forget gate), NOAF (no output tanh) hurts.
- **Jozefowicz et al.**: 10,000 runs; bias of 1 for LSTM's forget gate ≈ GRU. Did not improve on LSTM



Greff et al. 2015/17; Jozefowicz, Zaremba & Sutskever 2015

Many empirical studies of RNNs/LSTMs 2015–2017

Many early attempts to improve on RNNs/LSTMs via large scale search

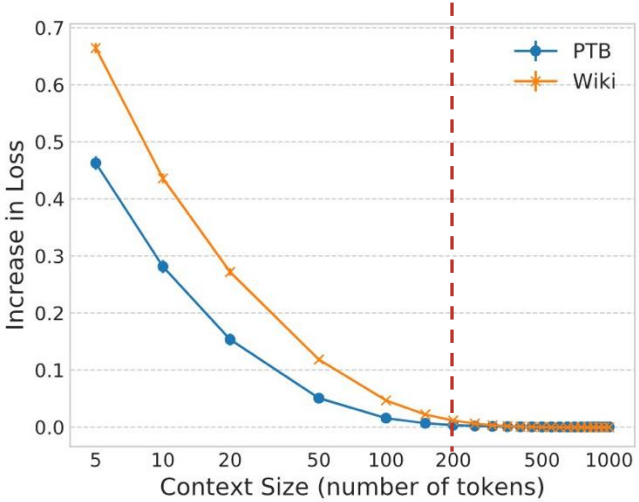
General findings – GRU/LSTM is about the best one can find after tuning

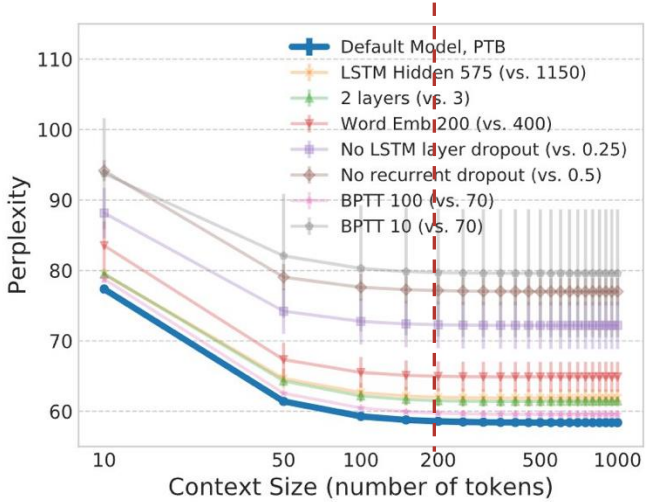
Study	Scale	Lesson
Jozefowicz et al. 2015	10,000 runs	search rediscovers the GRU; forget-gate bias of 1
Greff et al. 2015/17	5,400 runs	tune the learning rate first; the rest barely interacts
Britz et al. 2017	250k GPU-hours	report seed variance; deep decoders need residuals
Melis et al. 2017	large tuning budget	a tuned LSTM beats the newer cells

How much context does a LSTM LM use in practice?

Ablations suggested LSTM context use was ~ 200 tokens

- Word order matters only within the last ~50 tokens





Khandelwal et al. 2018; Miller & Hardt 2019

Solving propagation (at a cost)

Published as a conference paper at ICLR 2015

NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

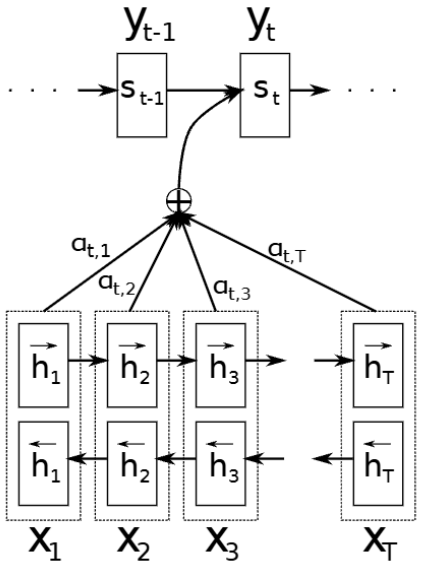
KyungHyun Cho **Yoshua Bengio***
Université de Montréal

We can avoid all these hoops and form more direct paths

The solution for temporal dependence is different than depth

- We aren't *inherently* looking for sequential processing
- Can directly connect our inputs to outputs
- Use per-position, input dependent gates to decide mixing

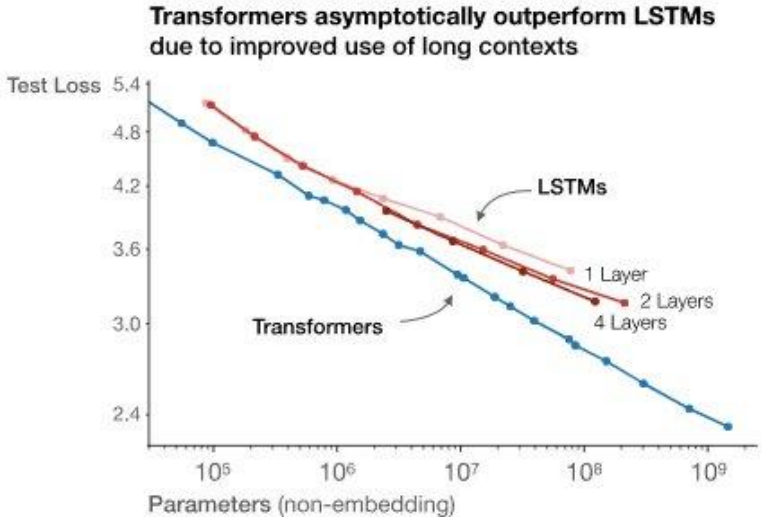
This is attention. Here, we want shallow, not deep

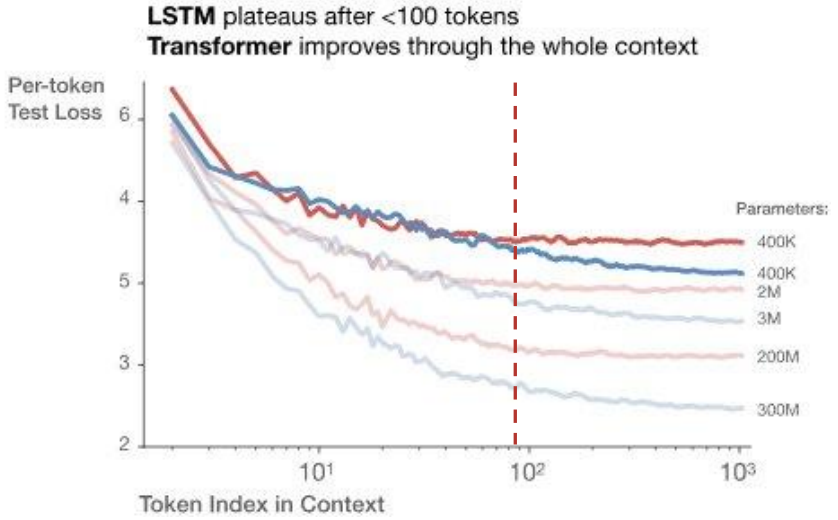


Significant improvements in scaling

Much better scaling properties for attention based models (like transformers)

- Scaling with larger models
- Scaling with longer context beyond 100-200 tokens.





Kaplan et al. 2020

Summary

Retention is one dimensional signal propagation over time

The LSTM is a similar solution to residuals. Straight signal path in time

But much like residuals, this doesn’t guarantee learnability of the units

Attention leverages the fact that we don’t need to actually have ‘depth’

Lecture today

Part 1

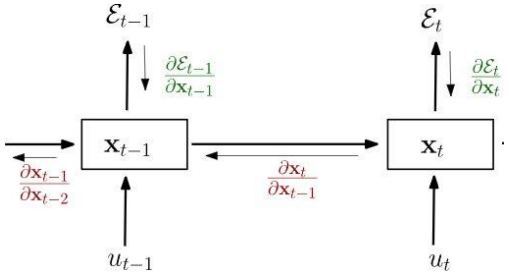
Retention: RNNs to LSTMs

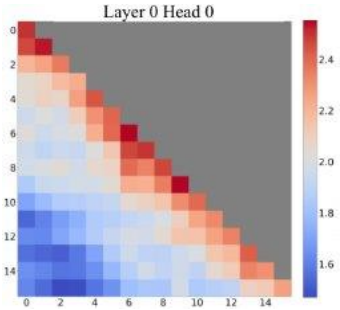
Part 2

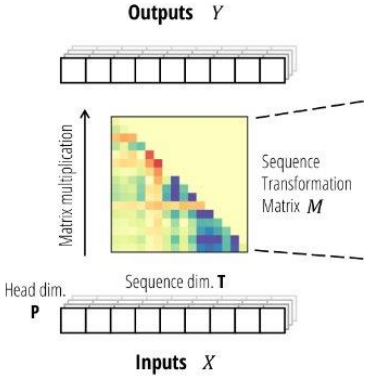
Mixing: attention

Part 3

Capacity: SSMs vs LSTMs







Attention: direct access, convex weights

Attention is a 'gated' connection from any input to our current position

$$y_t = \sum_{s \le t} A_{ts} v_s, \qquad A_{ts} = \frac{e^{q_t \cdot k_s / \sqrt{d}}}{\sum_{r \le t} e^{q_t \cdot k_r / \sqrt{d}}}, \qquad \sum_{s \le t} A_{ts} = 1, \qquad A_{ts} \ge 0$$

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

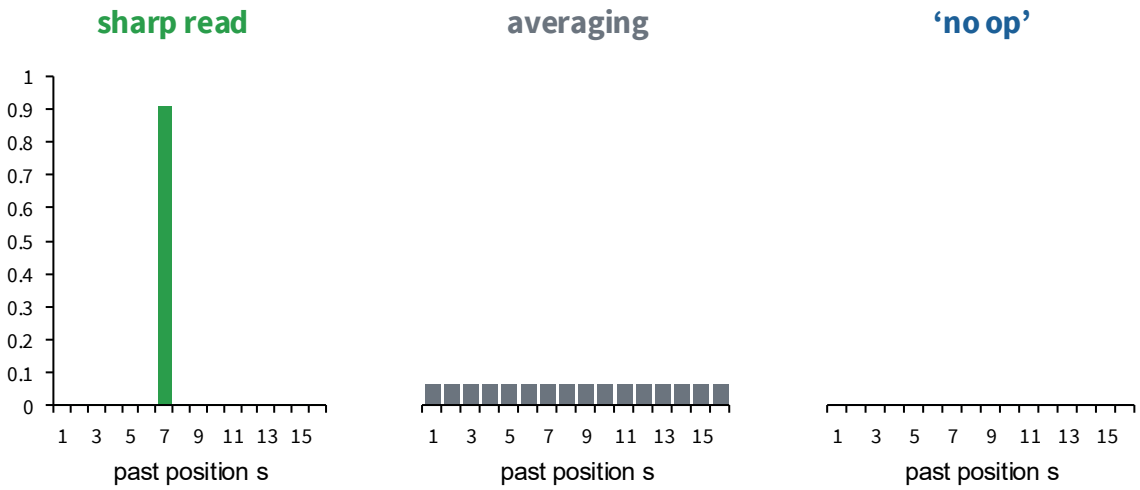
This has significant advantages (path length) but also complexities (what should the weights be?)

Vaswani et al. 2017, Table 1; Bahdanau, Cho & Bengio 2014

What do we want to mix?

We might want to do many different kinds of operations..

- **Sharp retrieval:** needle-in-the-haystack type find and retrieve tasks
- **Averaging**: identify topic or other diffuse, distributional information
- **No op:** no temporal information wanted or needed (in upper layers)

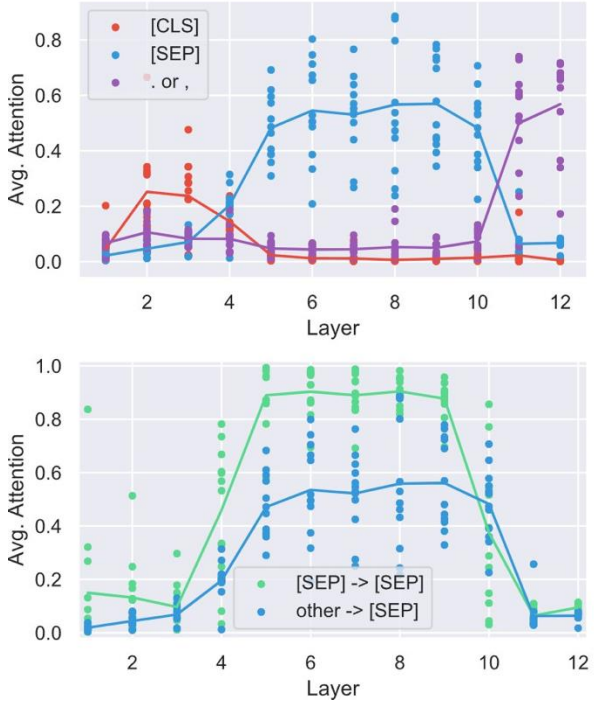


What are attention heads looking at? (2019)

Early study on attention head weights for BERT in 2019

Many heads put most of their weight on the sentence separator [SEP], a ‘meaningless’ token

- Over half of all attention in the middle layers goes to [SEP]
- No meaningful gradient activity for these layers either

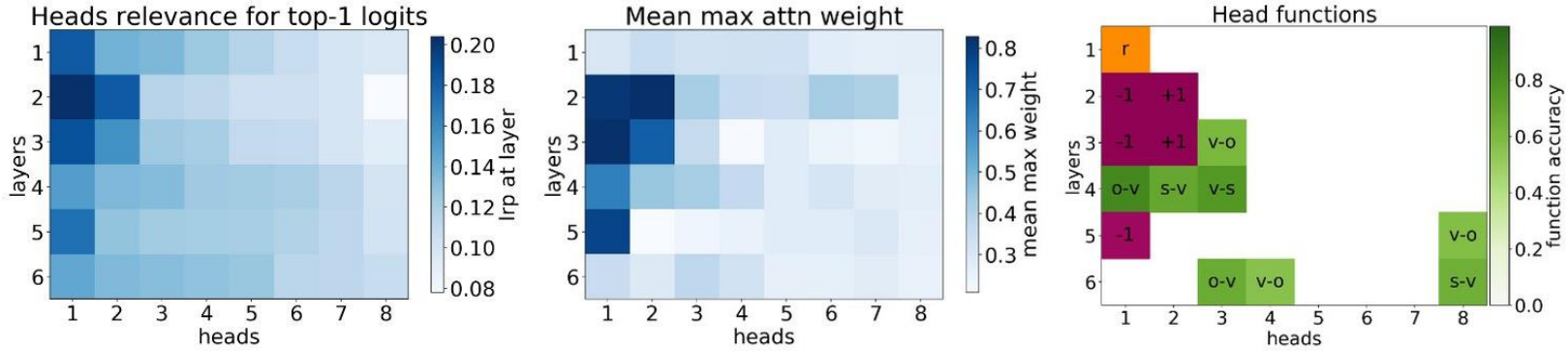


Clark, Khandelwal, Levy & Manning 2019

Heads are redundant in a translation model (2019)

Removing 38 / 48 attention heads decreases performance by only 0.15 BLEU

- The remaining 10/48 have ‘interpretable’ roles like tracking position.
- Others have found very similar phenomena (e.g. Michel, Levy and Neubig 2019)

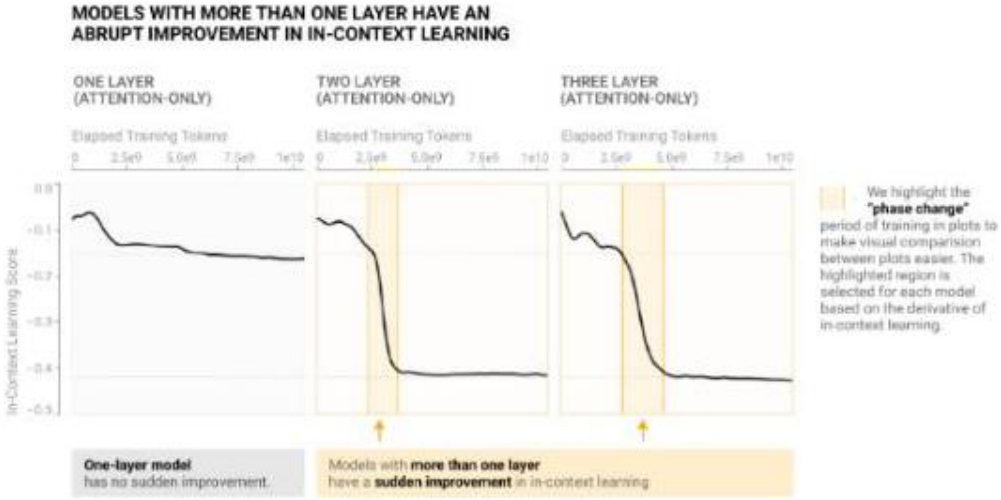


Voita et al. 2019

But some heads do very important work..

‘Induction heads’ enable copy-and-track behaviors for past in-context tokens

- Good case of attention’s ability to precisely track past tokens
- Requires scale (multiple layers, FLOPs) to learn.

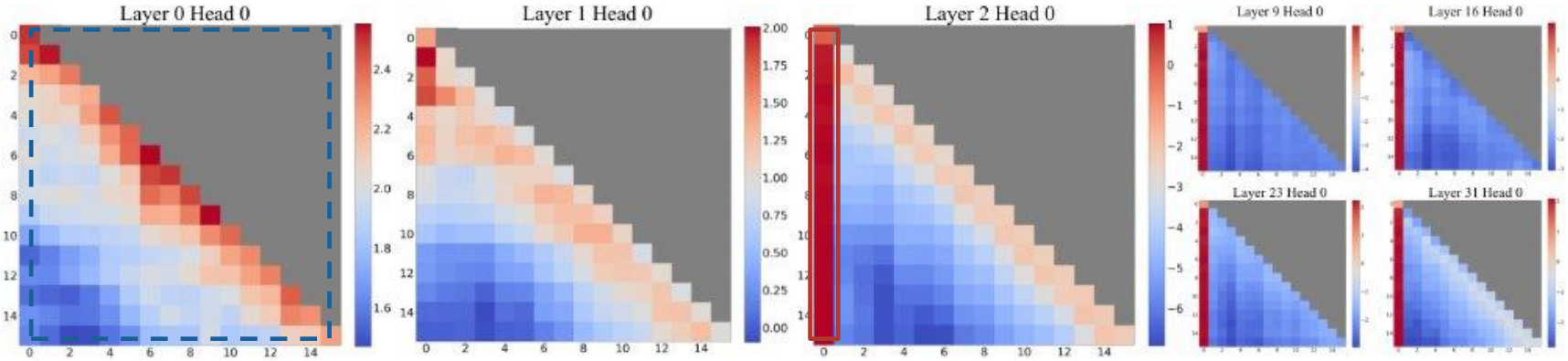


Olsson et al. 2022

Attention sink (2023)

Beyond the first two layers, most heads in LLMs attend heavily to the first token

- The first token's value is ≈ 0: attending to it adds nothing
- What is this doing? Reminiscent of our [SEP] token behavior: strong attention on useless token

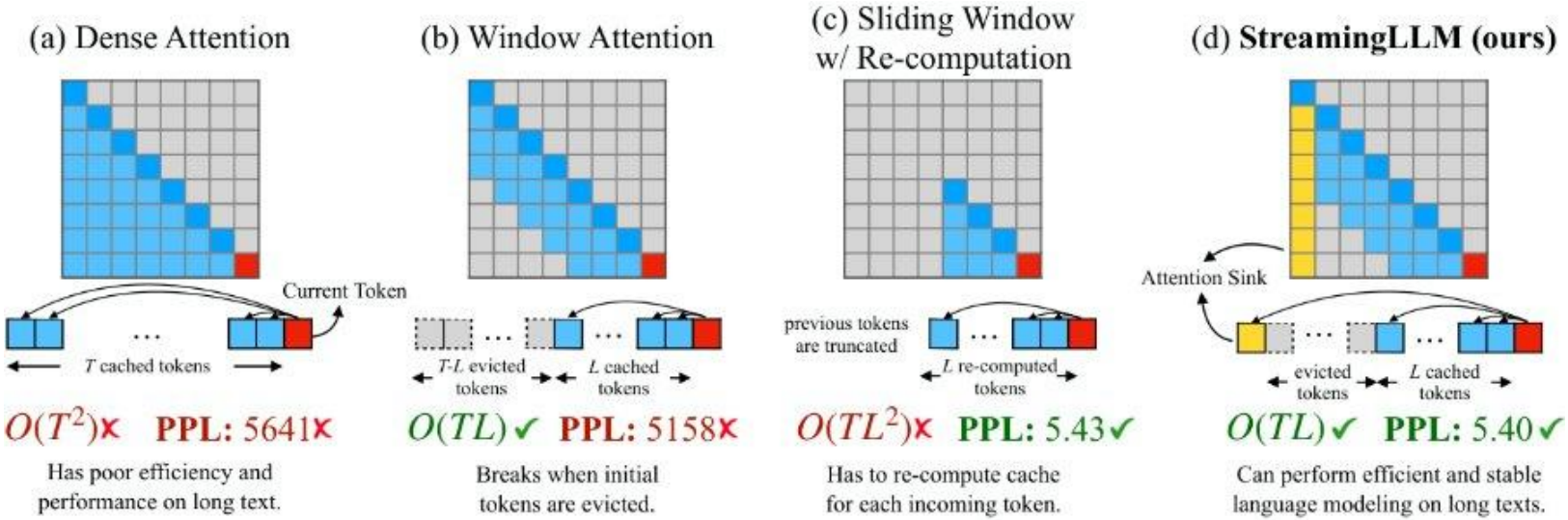


Xiao, Tian, Chen, Han & Lewis 2023

StreamingLLM: implications for KV cache eviction

When doing sliding windows, evicting the first-token KV cache degrades model performance

- Perplexity 5,158 without the initial attention sink token
- Instead, can train with an explicit sink token (or we can gate attention)

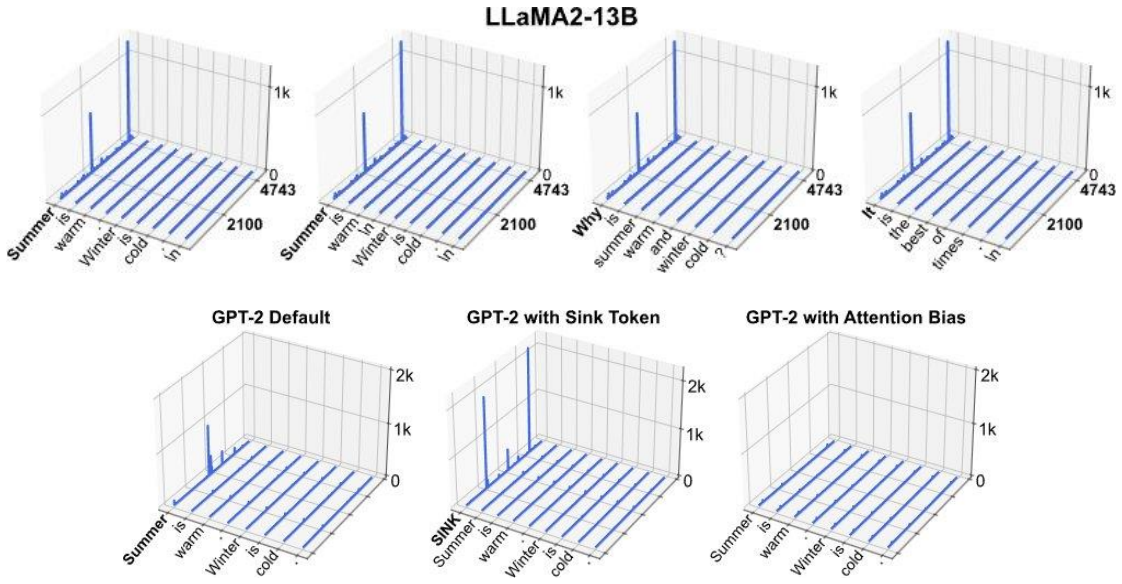


Xiao et al. 2023

Side effects and causes of the sink

Observation: models can have massive activations on the first token

- Massive activations erase residual contributions and makes a ‘universal key’
- If you add an explicit key/value bias, they disappear



Sun, Chen, Kolter & Liu 2024

Ablations on the sum to one constraint

Ablations on attention style mixing mechanisms

softmax: $\frac{e^{s_{ts}}}{\sum_r e^{s_{tr}}}$ sigmoid, normalised: $\frac{\sigma(s_{ts})}{\sum_r \sigma(s_{tr})}$ sigmoid, no sum: $\sigma(s_{ts})$ (row may sum to anything)

Attention operation	Sink on token 1	Valid loss
softmax	18.2%	3.73
sigmoid, still divided by the sum	30.2%	3.74
sigmoid, no normalisation	0.4%	3.70
softmax, 1B parameters	45.1%	3.07
sigmoid, no normalisation, 1B	2.5%	3.10

Gu et al. 2025 (ICLR)

Varied solutions to attention sinks

There are many proposed fixes to let an attention ‘do nothing’

Method	Approach	Impact
Sink logits (gpt-oss)	denominator sum > 1	downscaled softmax
Learnable sink token	0 value slot for sink	sink token takes activations
Sigmoid attention	no normalization constraint	sink 0.4–2.5%, same loss
Attention Gating (Qwen3)	gating allows for no-op	sink activation 46.7% → 4.8%

Miller 2023; Xiao et al. 2023; Gu et al. 2025; Qiu et al. 2025

From attention to ‘space models’

Consider the usual attention operation.. $Q \in R^{n \times d_k}, K \in R^{n \times d_k}, V \in R^{n \times d_v}$

$$Attn(Q, K, V) = \rho(QK^\top)V$$

This is quadratic due to $QK^\top$ $(n^2 d_k)$. Can we do better (when ρ is the identity)?

$$(QK^\top)V = Q(K^\top V)$$

This is very silly, but surprisingly important.. We get from $n^2 d_k + n^2 d_v$ to $2nd_v d_k$

[Shen et al 2018, Katharopoulos 2020 for kernel version. Also connections to things like fast weight programmers]

Recurrent form of linear attention

Recall that in purely linear attention, we consider the reordering

$$(QK^\top)V = Q(K^\top V)$$

This is linear time (great) but the even nicer thing is that this looks like a RNN.

$$S_t = S_{t-1} + k_t v_t^\top \quad \text{and} \quad y_t = q_t^\top S_t$$

This ‘duality’ allows us to train efficiently (using the parallel, quadratic form) and inference efficiently (using the serial, linear form)

(Note: if one weights S_{t-1} by γ, you get a RetNet)

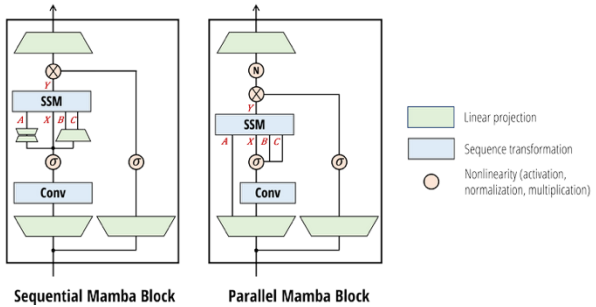
From linear attention to Mamba-2

Let's generalize linear attention a little bit and add per-position weights..

$$S_t = S_{t-1} + k_t v_t^\top \quad \text{and} \quad y_t = q_t^\top S_t \text{ (Linear attention)}$$
$$S_t = \gamma_t S_{t-1} + k_t v_t^\top \quad \text{and} \quad y_t = q_t^\top S_t + v_t^\top D \text{ with } \gamma_t = f(x_t) \text{ (Mamba-2)}$$

There is a lot more words to justify this (go read the mamba 2 paper) but the mechanics is that we can make linear attention more expressive via gating (gating is good!)

This continues to have duality properties (compute γ in parallel, apply duality)



Gated delta net (and friends)

Let's generalize things further – gate the input and selectively erase the state.

$$S_t = \gamma_t S_{t-1} + k_t v_t^\top \quad \text{and} \quad y_t = q_t^\top S_t + v_t^\top D \quad \text{with } \gamma_t = f(x_t) \text{ (Mamba-2)}$$

$$S_t = \gamma_t (I - \beta_t k_t k_t^\top) S_{t-1} + \beta_t k_t v_t^\top \quad \text{and}$$
$$y_t = q_t^\top S_t \text{ with } \gamma_t = f(x_t), \beta_t = f(x_t) \text{ (Gated Delta Net)}$$

The gated delta net adds a 'no input operation' gate ($\beta = 0$)

And *erases* anything in the direction of the current key ($I - \beta_t k_t k_t^\top$)

Close relationships to various fast weight programming / test time training ideas.

What happens with mixing in a recurrent model?

Mixing is always *per-step*.

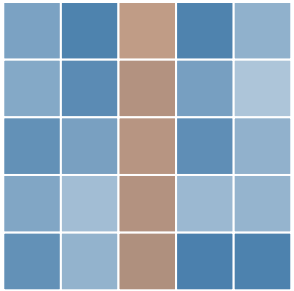
GRU / LSTM / Mamba



$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

Average each channel (GRU)

Linear attention

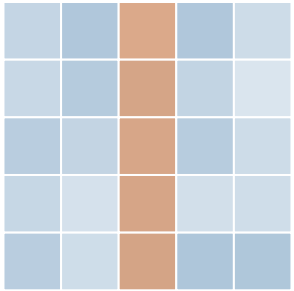


key k

$$S_t = S_{t-1} + v_t k_t^\top$$

Add

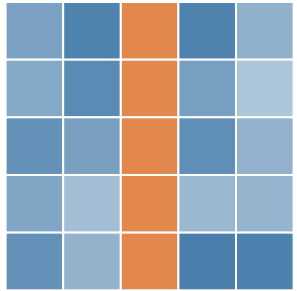
Mamba-2 / RetNet



$$S_t = \alpha_t S_{t-1} + v_t k_t^\top$$

Forget then add

DeltaNet



$$S_t = S_{t-1}(I - \beta_t k_t k_t^\top) + \beta_t v_t k_t^\top$$

Forget in the 'k' subspace

old content new input both, superposed

Cho et al. 2014; Katharopoulos et al. 2020; Dao & Gu 2024; Schlag et al. 2021

A closer look at the Gated DeltaNet

All recurrent architectures have hard tradeoffs in forget vs store

- **Mamba2** (decay only): falls off at 8K NIAH due to storage tradeoffs
- **DeltaNet** (delta only): forced to retain at each step, so fails on long, noisy content (essays)
- **Gated DeltaNet:** allows for gates to zero out the memory (forgetting), widely adopted

$$S_t = S_{t-1}(\alpha_t(I - \beta_t k_t k_t^\top)) + \beta_t v_t k_t^\top$$

	passkey in a synthetic haystack				number in essays				UUID in essays		
1.3B models	**1K**	**2K**	**4K**	**8K**	**1K**	**2K**	**4K**	**8K**	**1K**	**2K**	**4K**
Mamba2	99.2	98.8	65.4	30.4	99.4	98.8	56.2	17.0	64.4	47.6	4.6
DeltaNet	97.4	96.8	99.0	98.8	98.4	45.6	18.6	14.4	85.2	47.0	22.4
Gated DeltaNet	98.4	88.4	91.4	91.8	100	99.8	92.2	29.6	86.6	84.2	27.6

Mixing across different architecture choices

Back to our high level view

	No-op	Averaging	Sharp retrieval
Attention	rows sum to one: needs a sink	Uniform attention is averaging	Large KV inner product allows sharp retrieval
Gated recurrence (GRU, LSTM, Mamba)	gated inputs ($z = 0, \alpha = 1$) leave state the same	can express EMA-style averaging	erasing a new token (t) erases old tokens at (s<t)
Delta recurrence (DeltaNet, Gated DeltaNet)	gated inputs ($\beta = 0, \alpha = 1$)	EMA only with a decay gate (Gated DeltaNet)	d linearly independent items can be stored

- Different tradeoffs – even more if you add the systems concerns for attention
- Some operations are natural in the ‘sequential’ view of RNN/LSTM/SSMs
- Capacity remains a concern for these sequential models.

Summary

Dealing with time is more complicated than simple signal propagation

Have to figure out what information to get, how to mix it

Tradeoffs: Attention is direct, but complex, global mixing. SSMs are simple but the mixing and information propagation is harder to reason about

- Leads to issues like attention sinks
- SSMs on the other hand, require complex updates to make work

Ideally, we want the capacity of attention without its cost.

Lecture today

Part 1

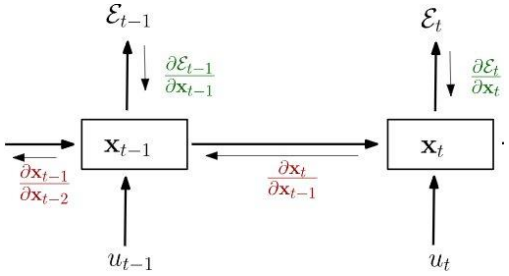
Retention: RNNs to LSTMs

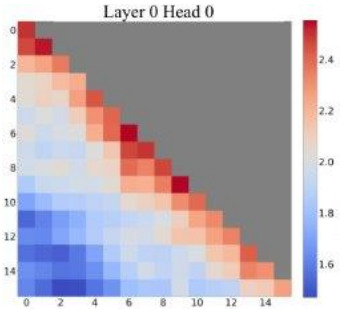
Part 2

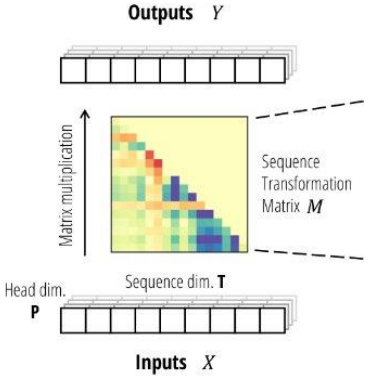
Mixing: attention

Part 3

Capacity: SSMs vs LSTMs







Tradeoffs among our model classes

What is wrong with each of our existing sequence mixing layers?

Attention

Pro: Easy, direct signal propagation. Unlimited, precise recall

Con: Systems costs (quadratic cost). Mixing may be complex (positions, sinks)

LSTMs / SSMs

Pro: Linear time, simple mixing that works for local information

Con: Long range capacity is bounded, training can be complex

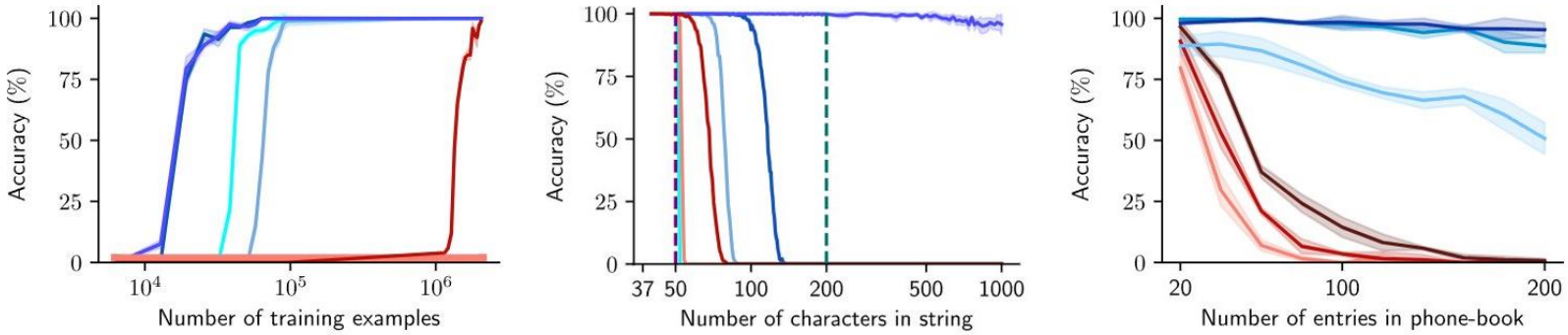
What dictates the capacity limits of bounded capacity sequence mixers?

Limitations for state-based models (LSTMs, SSMs)

Transformers learn to copy from far fewer examples, and generalise to longer strings

- A fixed-size state must compress the input; a KV cache keeps every token
- Pretrained Mamba loses at copying even when its perplexity matches

Blue: Transformer
Red: SSM

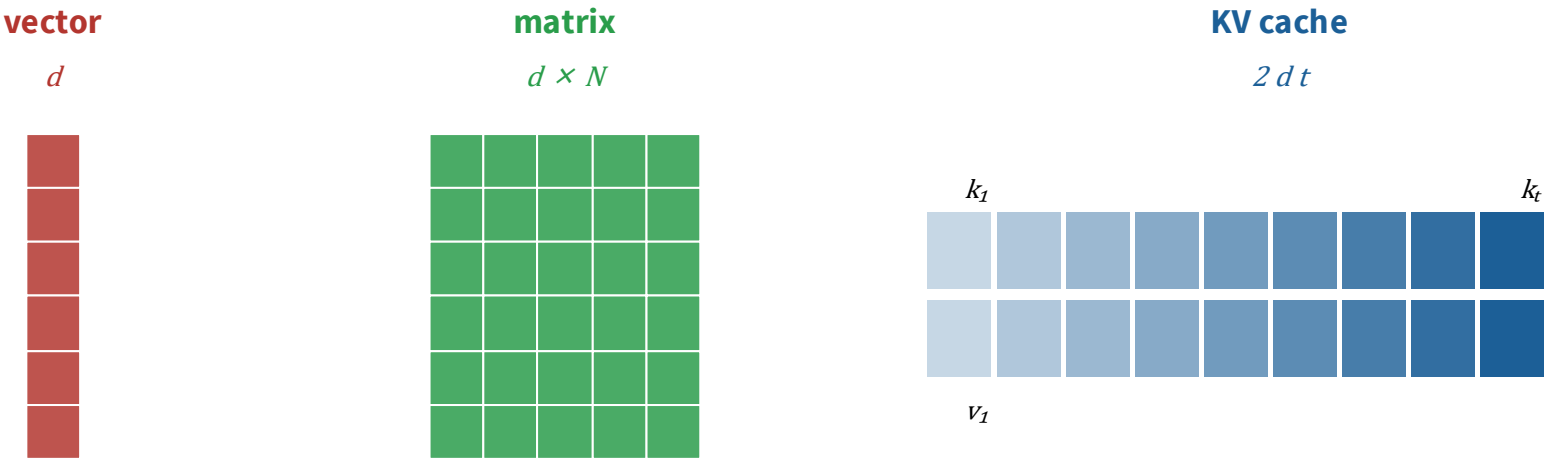


Jelassi, Brandfonbrener, Kakade & Malach 2024

Capacity for different types of state

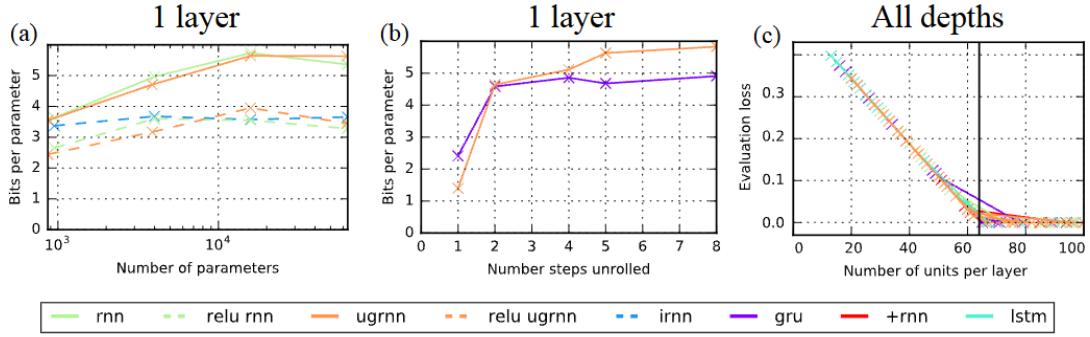
How much information can we carry across time?

- **Vector** (RNN, LSTM): d scalars. Increasing the state has *quadratic* compute cost.
- **Matrix** (linear attention, DeltaNet, Mamba-2): d × N matrix. State size N can grow independently
- **KV cache** (attention): State grows linearly in time (quadratic in time total cost)

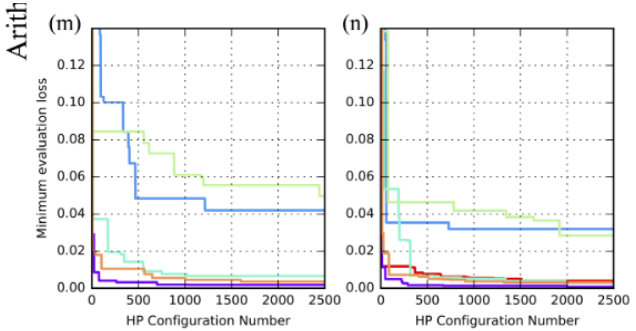


What is the capacity of a LSTM? (2017)

LSTM variants all can ‘memorize’ similar amounts of information (1 number / unit)



Main differences they find are in the trainability of more modern LSTM variants



Collins, Sohl-Dickstein & Sussillo 2017

A basic view of linear memory and readouts

Store key–value pairs, read back with a query: recall = signal + interference from the others

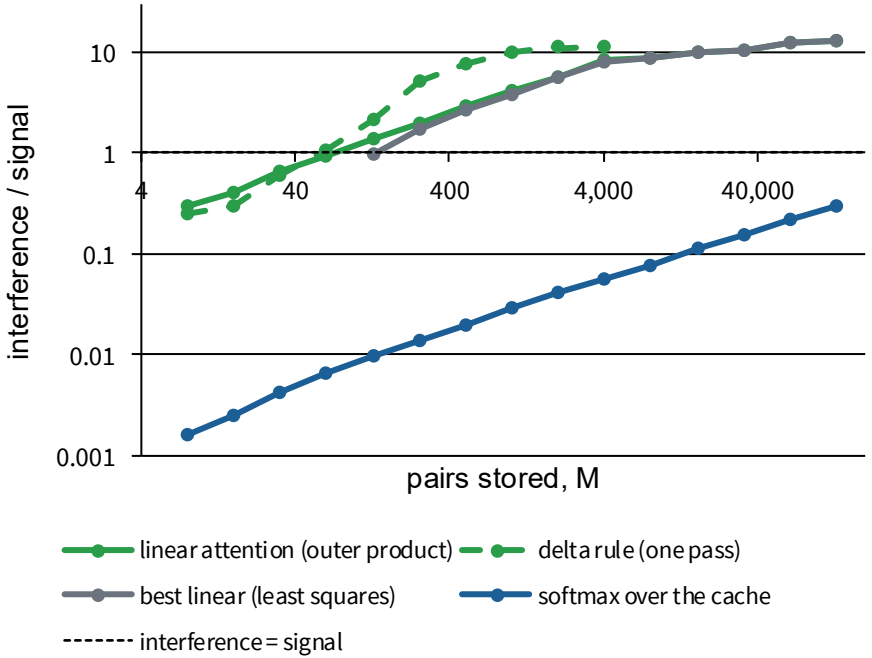
$$S = \sum_i v_i k_i^\top: \quad Sk_j = v_j(k_j^\top k_j) + \sum_{i \neq j} v_i(k_i^\top k_j) \quad \text{(signal + interference)}$$

Store	Write	Read with query q	Where it shows up
Outer product (Hebbian)	$S \leftarrow S + v\,k^\top$	$S\,q$	linear attention
Delta rule	$S \leftarrow S - \beta\,(S\,k - v)\,k^\top$	$S\,q$	DeltaNet, Gated DeltaNet
Best linear (least squares)	$S = V\,K^+$	$S\,q$	an ‘idealized’ delta rule
KV cache + softmax	append (k, v)	$\Sigma_i \text{softmax}(\beta\, k_i^\top q)\, v_i$	attention
MLP	learned W_1 (keys), W_2 (values)	$W_2\,\sigma(W_1\,q)$	feed-forward layers

Wang 2026 (blog, ‘You could have invented transformers from memory’); Schlag et al. 2021; Geva et al. 2021

How many memories fit?

Random unit keys and values, $d_k = 64$: store M pairs, then read each one back



Store	Pairs that fit
Outer product	$\approx d_k = 64$
Best linear	$\approx 2d_k = 128$
ReLU MLP	$\approx 2d_k = 128$
Squared-ReLU MLP	$\approx 2d_k^2/3 \approx 2{,}700$
Cubed-ReLU MLP	$\approx 2d_k^3/15 \approx 35{,}000$
Softmax, $\beta = \sqrt{d_k}$	$\approx e^{14} \approx 1.2$ M

Pairs that fit = M where interference = signal.
More nonlinear read-outs separate far more keys.

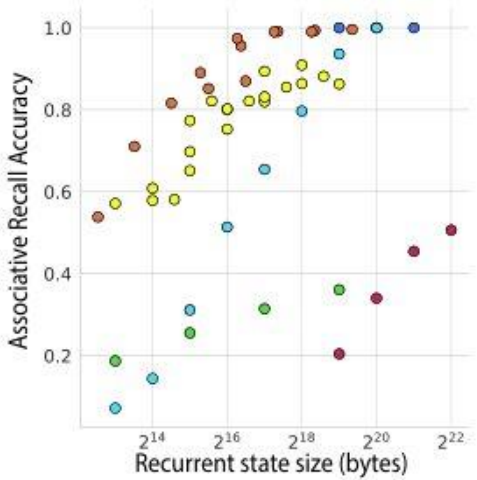
Setup from Wang 2026 (blog)

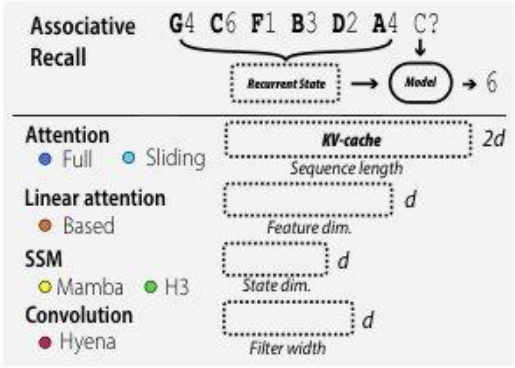
A modern take on probing-based capacity measures

A synthetic task (MQAR) where varying state size changes the overall accuracy

- **Task** (bottom of figure): a context of key–value pairs, then queries for several keys
- **How do models do?**
  - **Attention** is at the top (high size, high acc)
  - **Sliding window** has unfavorable tradeoffs
  - **Various SSMs can push the pareto frontier**

In principle, a nice analysis, but highly dependent on the toy task. Also often over-optimized for.



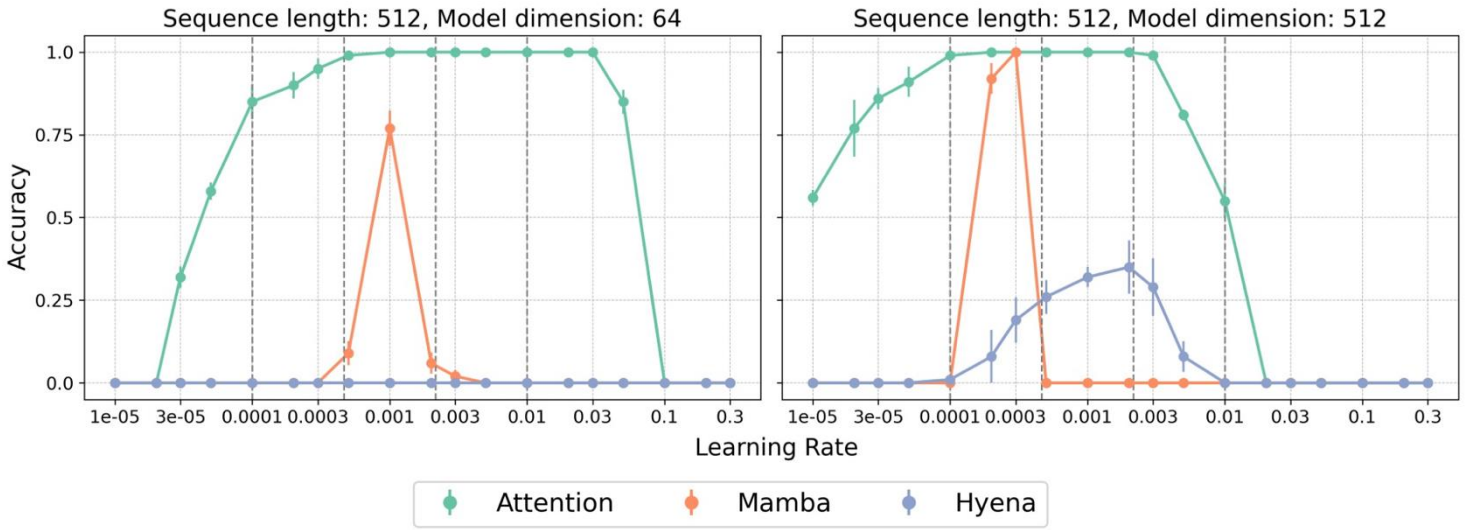


Arora et al. 2024 (Based), Fig. 2; Arora et al. 2023 (Zoology)

Pitfalls in probe based capacity measures

MQAR accuracy vs learning rate (length 512): attention trains almost anywhere; Mamba and Hyena in a narrow window

- **Dashed lines**: a typical benchmark learning-rate grid, coarse enough to miss Mamba's window
- **Width vs depth**: recurrent models gain from width (more state), attention from depth

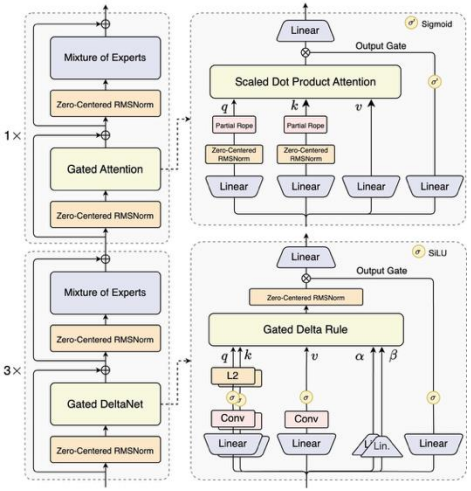


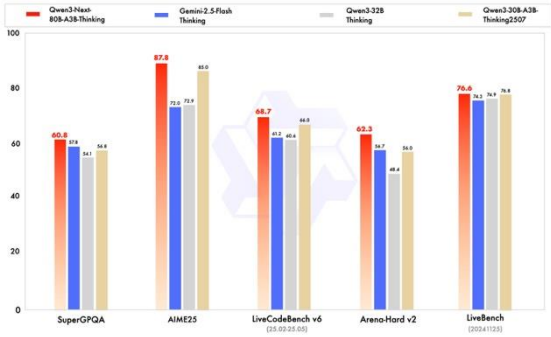
Okpekpe & Orvieto 2025, Fig. 1

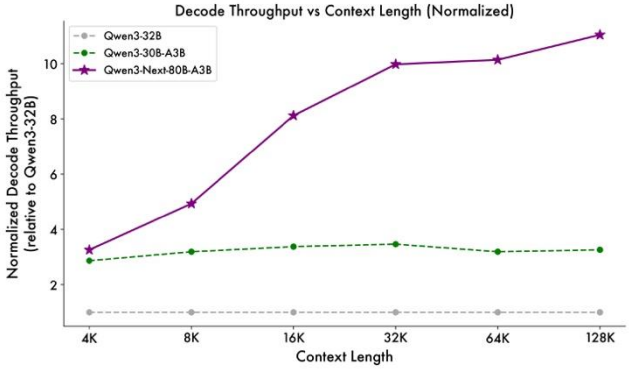
Summary

- **Basic memory accounting:** a N-element state holds at most N numbers
- **Readout constraints:** but linear readouts can significantly underuse this state!
- **Empirical probing measurements are important**
  - But also major pitfalls like training the probe models
  - Or overfitting on the toy tasks

Sequence mixing in practice – Qwen Next







The newest qwen are 3-1 GDN / Attention hybrids.

Pretty reasonable performance with good inference characteristics

Hybrid performance

ByteDance | Seed UC SANTA CRUZ

A Systematic Analysis of Hybrid Linear Attention

Dustin Wang*[1], Rui-Jie Zhu*[1,2],
Steven Abreu[3], Yong Shan[2], Taylor Kergan[1], Yuqi Pan[2,4], Yuhong Chou[5], Zheng Li[2],
Ge Zhang[2,6,†], Wenhao Huang[2], Jason Eshraghian[1,†]

[1]UC Santa Cruz, [2]ByteDance Seed, [3]University of Groningen, [4]CASIA, [5]PolyU, [6]M-A-P

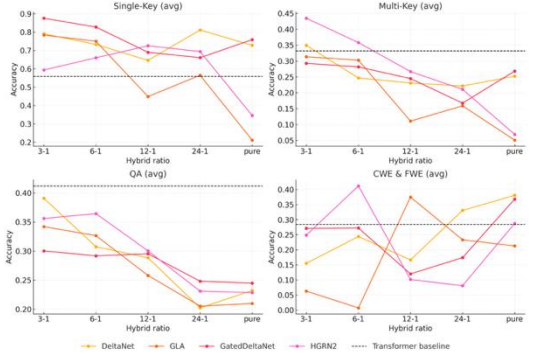
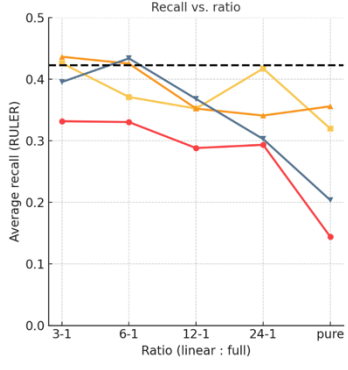


Figure 4 RULER sub task results based on ratio. RetNet and HGRN model families are omitted as their recall benchmark results were insignificant.

Model	Update rule ($\mathbf{S}_t$)	Read-out (o_t)
Vector-valued hidden state (classical / gated RNNs)		
HGRN[4]	$\boldsymbol{h}_t = \boldsymbol{\alpha}_t \odot \boldsymbol{h}_{t-1} + (1-\boldsymbol{\alpha}_t) \odot \boldsymbol{v}_t$	$\boldsymbol{o}_t = \boldsymbol{h}_t \odot \boldsymbol{q}_t$
Hawk (RG-LRU)[33]	$\boldsymbol{h}_t = \boldsymbol{r}_t \, \boldsymbol{h}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{x}_t$	$\boldsymbol{o}_t = \boldsymbol{h}_t \odot \boldsymbol{q}_t$
Matrix-valued state via outer products		
RetNet/Lightning [11, 17]	$\mathbf{S}_t = \gamma \mathbf{S}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^\top$	$\boldsymbol{o}_t = \mathbf{S}_t \boldsymbol{q}_t$
GLA[34]	$\mathbf{S}_t = \mathbf{S}_{t-1} \odot (\mathbf{1}\boldsymbol{\alpha}_t^\top) + \boldsymbol{v}_t \boldsymbol{k}_t^\top$	$\boldsymbol{o}_t = \mathbf{S}_t \boldsymbol{q}_t$
Mamba-2[35]	$\mathbf{S}_t = \gamma_t \, \mathbf{S}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^\top$	$\boldsymbol{o}_t = \mathbf{S}_t \boldsymbol{q}_t$
RWKV-6[36]	$\mathbf{S}_t = \mathbf{S}_{t-1} \mathrm{Diag}(\boldsymbol{\alpha}_t) + \boldsymbol{v}_t \boldsymbol{k}_t^\top$	$\boldsymbol{o}_t = (\mathbf{S}_{t-1} + (\boldsymbol{d} \odot \boldsymbol{v}_t)\boldsymbol{k}_t^\top)\boldsymbol{q}_t$
HGRN-2/MetaLA[4, 37]	$\mathbf{S}_t = \mathbf{S}_{t-1} \mathrm{Diag}(\boldsymbol{\alpha}_t) + \boldsymbol{v}_t (1-\boldsymbol{\alpha}_t)^\top$	$\boldsymbol{o}_t = \mathbf{S}_t \boldsymbol{q}_t$
Delta-rule / controlled-forgetting family		
DeltaNet[5]	$\mathbf{S}_t = \mathbf{S}_{t-1}(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top) + \beta_t \boldsymbol{v}_t \boldsymbol{k}_t^\top$	$\boldsymbol{o}_t = \mathbf{S}_t \boldsymbol{q}_t$
Gated DeltaNet[19]	$\mathbf{S}_t = \alpha_t \, \mathbf{S}_{t-1}(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^\top) + \beta_t \boldsymbol{v}_t \boldsymbol{k}_t^\top$	$\boldsymbol{o}_t = \mathbf{S}_t \boldsymbol{q}_t$



Not many controlled ablations, but some evidence of low losses at small hybrid ratios

Recap

Similar to last time: Signal propagation and trainability in RNN/LSTMs

Significant differences: Time mixing can be done w/ short, direct connections

Main concerns are systems-architecture tradeoffs

- Attention like trainability and memorization without the cost
- State based models with higher capacity

These ideas are still major works in progress!