Memory and Forgetting

Getting to Continual Learning

Transformers, redux

The goal was just to get rid of sequential state updates so everything could train in parallel.

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

V = the information from the past ("book contents").
Q = the question we want to answer right now ("question").
K = the description of each piece of the past ("book description").

QK^T = "which books will have relevant content to answer my current question?"

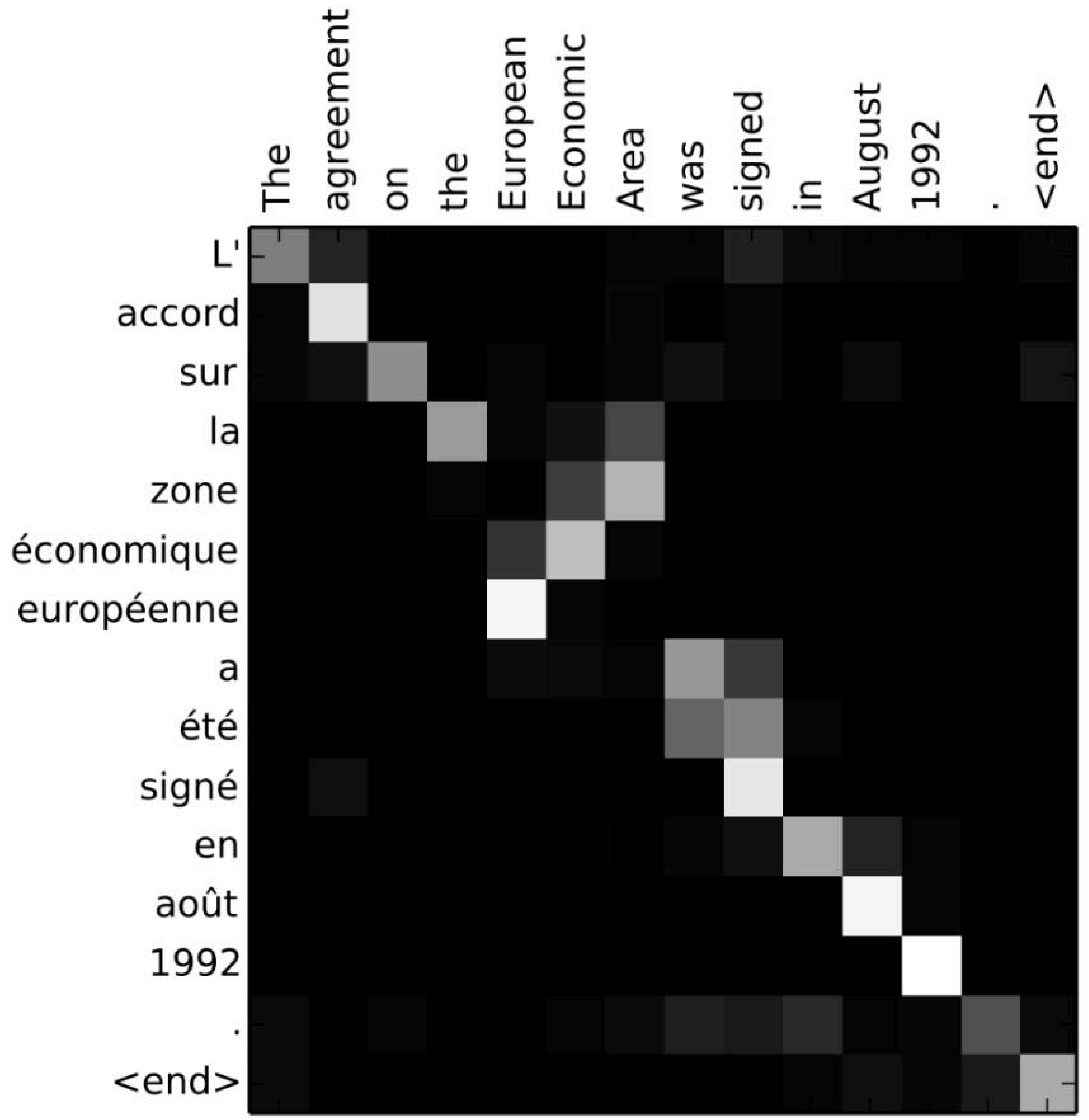
softmax = "turn it into proportions / relevance scores from 0 to 1 that sum to 1."

QK * V = linear interpolation of book contents weighted by relevance.

Sequence Modeling

but we end up with quadratic time/space complexity wrt sequence length: each token attends to every other token in the sequence. (this is QK^T)

“basically just keep storing summaries of the past every time” → KV caching



Attention-matrix heatmap

Bahdanau, et al. 2015. Neural machine translation by jointly learning to align and translate. In Proc. ICLR.

grows O(n) with sequence length

grows O(n) with sequence length

O(n^2) memory

so we want to improve upon that to *subquadratic* complexity.

Various Attempts to Be Smart About Quadratic Attention

Grouped Query Attention

Sparse Attention

Latent Attention

Linear Attention

The crux of linear attention is that we substitute the softmax with a linearized kernel.

$$O_i = \frac{\phi(Q_i)^T \sum_{j=1}^{i}(\phi(K_j)V_j^T)}{\phi(Q_i)^T \sum_{j=1}^{i}\phi(K_j)}$$

$$S_i = \sum_{j=1}^{i} \phi(K_j)V_j^T$$

$$S_i = \sum_{j=1}^{i-1} \phi(K_j)V_j^T + \phi(K_i)V_i^T = S_{i-1} + \phi(K_i)V_i^T$$

By writing the softmax explicitly, Eq. 7 can be written as:

$$\boldsymbol{y}^{(i)} = \sum_{j=1}^{i} \frac{\boldsymbol{v}^{(j)}\kappa(\boldsymbol{k}^{(j)}, \boldsymbol{q}^{(i)})}{\sum_{j'=1}^{i}\kappa(\boldsymbol{k}^{(j')}, \boldsymbol{q}^{(i)})} \tag{12}$$

where $\kappa(\boldsymbol{k}, \boldsymbol{q}) = \exp(\boldsymbol{k} \cdot \boldsymbol{q}) \in \mathbb{R}_{>0}$ is the softmax kernel and $\boldsymbol{k} \cdot \boldsymbol{q} = \boldsymbol{k}^\top \boldsymbol{q}$ is the vector dot product.

The general idea is to replace the softmax kernel κ by another kernel: $\kappa'(\boldsymbol{k}, \boldsymbol{q}) = \phi(\boldsymbol{k})^\top \phi(\boldsymbol{q})$ where ϕ is a function $\mathbb{R}^{d_{\text{key}}} \to \mathbb{R}^{d_{\text{dot}}}$. We discuss the necessary properties of ϕ in Sec. 5.1. By replacing κ in Eq. 12 by κ', we obtain

$$\boldsymbol{y}^{(i)} = \sum_{j=1}^{i} \frac{\boldsymbol{v}^{(j)}\phi(\boldsymbol{k}^{(j)})^\top \phi(\boldsymbol{q}^{(i)})}{\sum_{j'=1}^{i}\phi(\boldsymbol{k}^{(j')}) \cdot \phi(\boldsymbol{q}^{(i)})} \tag{13}$$

$$= \frac{\sum_{j=1}^{i}\left(\boldsymbol{v}^{(j)}\phi(\boldsymbol{k}^{(j)})^\top\right)\phi(\boldsymbol{q}^{(i)})}{\left(\sum_{j'=1}^{i}\phi(\boldsymbol{k}^{(j')})\right) \cdot \phi(\boldsymbol{q}^{(i)})} \tag{14}$$

This is an RNN

linear attention

“each new token interacts with the accumulated state”

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^\intercal \in \mathbb{R}^{d_v \times d_k}$$

O(1) space **O(n) time**

quadratic in *feature dimension*

quadratic in *sequence length*

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

O(n) space **O(n^2) time**

“each new token interacts with all previous tokens”

vanilla attention

since state is constant size, forgetting is important

Forget Gates (2000)
learn how to forget accumulated state

LSTMs (1997)
accumulate too much state, can’t solve repetitive FSA grammars

https://sferics.idsia.ch/pub/juergen/FgGates-NC.pdf

Linear Transformers = Fast Weight Programmers

When we linearize attention, it turns out it's equivalent to a Fast Weight Programmer.

$$\boldsymbol{k}^{(i)}, \boldsymbol{v}^{(i)}, \boldsymbol{q}^{(i)} = \boldsymbol{W}_k \boldsymbol{x}^{(i)}, \boldsymbol{W}_v \boldsymbol{x}^{(i)}, \boldsymbol{W}_q \boldsymbol{x}^{(i)} \tag{4}$$

$$\boldsymbol{W}^{(i)} = \boldsymbol{W}^{(i-1)} + \boldsymbol{v}^{(i)} \otimes \phi(\boldsymbol{k}^{(i)}) \tag{17}$$

$$\boldsymbol{z}^{(i)} = \boldsymbol{z}^{(i-1)} + \phi(\boldsymbol{k}^{(i)}) \tag{18}$$

$$\boldsymbol{y}^{(i)} = \frac{1}{\boldsymbol{z}^{(i)} \cdot \phi(\boldsymbol{q}^{(i)})} \boldsymbol{W}^{(i)} \phi(\boldsymbol{q}^{(i)}) \tag{19}$$

which is a Fast Weight Programmer (Sec. 2) with normalisation. Hence, the core of linear Transformer variants are outer product-based Fast Weight Programmers.

https://proceedings.mlr.press/v139/schlag21a/schlag21a.pdf

Fast Weight Programmers

Idea: a recurrent sequence model that can dynamically update its weights based on input.

"slow weights"

$$\boldsymbol{a}^{(i)}, \boldsymbol{b}^{(i)} = \boldsymbol{W}_a \boldsymbol{x}^{(i)}, \boldsymbol{W}_b \boldsymbol{x}^{(i)} \qquad (1)$$

$$\boldsymbol{W}^{(i)} = \sigma\big(\boldsymbol{W}^{(i-1)} + \boldsymbol{a}^{(i)} \otimes \boldsymbol{b}^{(i)}\big) \qquad (2)$$

$$\boldsymbol{y}^{(i)} = \boldsymbol{W}^{(i)} \boldsymbol{x}^{(i)} \qquad (3)$$

(1) "fast weights are dependent on input and slow weights"

(2) "write to short term memory with new fast weights"

(3) "retrieve the output sequence from the short term memory"

Write-only Access << Read-Write Access

if we can only write to the state, it will get clogged.
we want to be able to remove state that we think is wrong.

$$\bar{\boldsymbol{v}}^{(i)} = \boldsymbol{W}^{(i-1)}\phi(\boldsymbol{k}^{(i)}) \tag{20}$$

"read my previous state's attention to the current token"

$$\beta^{(i)} = \sigma(\boldsymbol{W}_\beta \boldsymbol{x}^{(i)}) \tag{21}$$

"how important is the current token"

$$\boldsymbol{v}_{\text{new}}^{(i)} = \beta^{(i)}\boldsymbol{v}^{(i)} + (1 - \beta^{(i)})\bar{\boldsymbol{v}}^{(i)} \tag{22}$$

"how much should I overwrite the state"

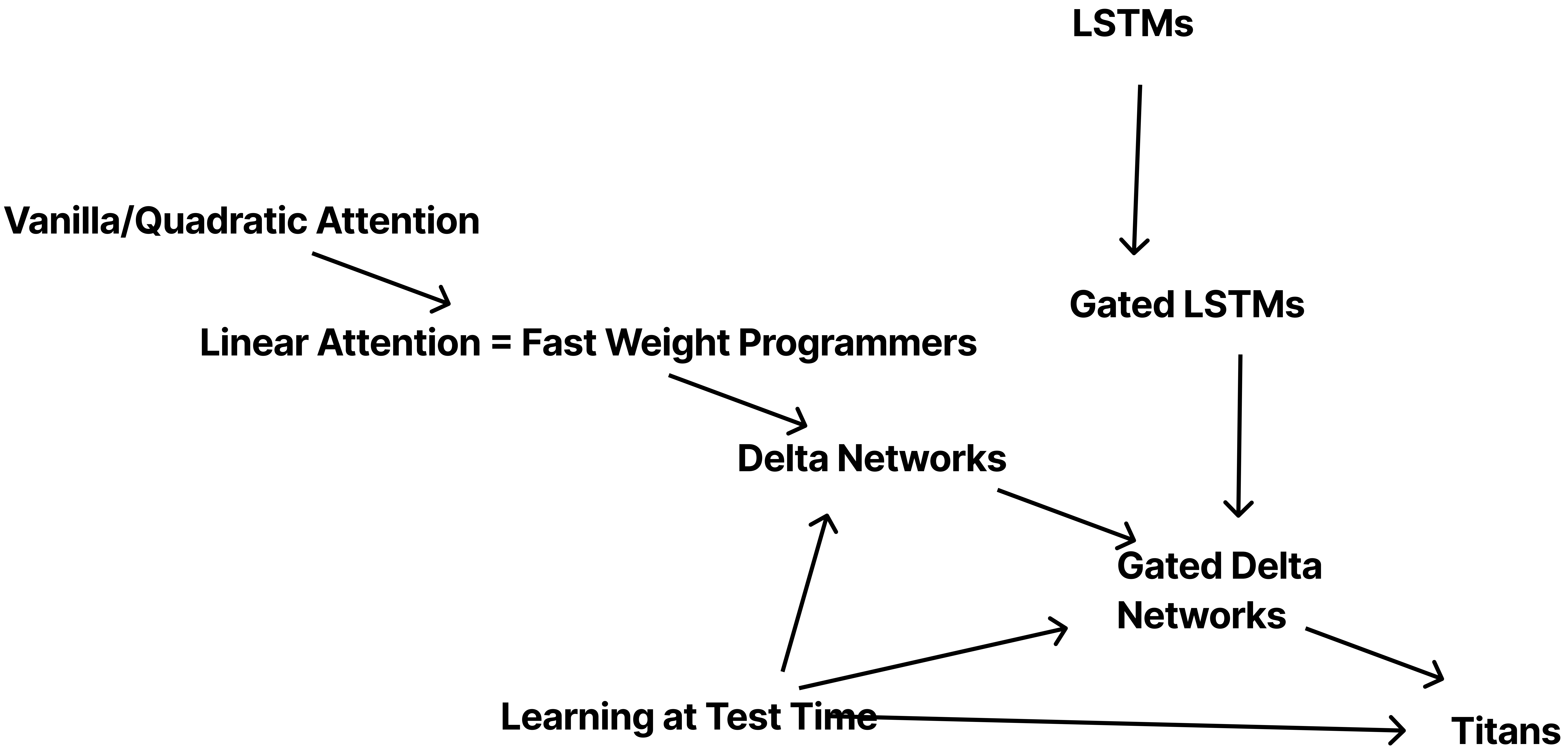
β = 0 → no update. β = 1 → total replacement.

$$\boldsymbol{W}^{(i)} = \boldsymbol{W}^{(i-1)} \underbrace{+\boldsymbol{v}_{\text{new}}^{(i)} \otimes \phi(\boldsymbol{k}^{(i)})}_{\text{write}} \underbrace{-\bar{\boldsymbol{v}}^{(i)} \otimes \phi(\boldsymbol{k}^{(i)})}_{\text{remove}} \tag{23}$$

$$= \boldsymbol{W}^{(i-1)} + \beta^{(i)}(\boldsymbol{v}^{(i)} - \bar{\boldsymbol{v}}^{(i)}) \otimes \phi(\boldsymbol{k}^{(i)}) \tag{24}$$

this is the *delta rule*

Associative Memory



Learning at Test Time

https://arxiv.org/pdf/2407.04620

To remain both efficient and expressive in long context, we need a better compression heuristic. Specifically, we need to compress thousands or potentially millions of tokens into a hidden state that can effectively capture their underlying structures and relationships. This might sound like a tall order, but all of us are actually already familiar with such a heuristic.

Online Update

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^T$$
$$\mathbf{S}_t = \alpha_t \mathbf{S}_{t-1} + \boldsymbol{v}_t \boldsymbol{k}_t^T$$
$$\mathbf{S}_t = \mathbf{S}_{t-1}(\mathbf{I} - \epsilon \boldsymbol{k}_t \boldsymbol{k}_t^T) + \epsilon_t \boldsymbol{v}_t \boldsymbol{k}_t^T, \epsilon_t = \frac{\beta_t}{1 + \beta_t \boldsymbol{k}_t^\top \boldsymbol{k}_t}$$
$$\mathbf{S}_t = \mathbf{S}_{t-1}(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^T) + \beta_t \boldsymbol{v}_t \boldsymbol{k}_t^T$$
$$\mathbf{S}_t = \mathbf{S}_{t-1}\left(\alpha_t(\mathbf{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^T)\right) + \beta_t \boldsymbol{v}_t \boldsymbol{k}_t^T$$

Objective

$J(S_t) = \|S_t - S_{t-1}\|_F^2 - 2\langle S_t k_t, v_t \rangle$

Gradient

$\nabla_{S_t} J = 2(S_t - S_{t-1}) - 2\, v_t k_t^\top$

Set to zero and solve

$2(S_t - S_{t-1}) - 2 v_t k_t^\top = 0 \;\Rightarrow\; S_t = S_{t-1} + v_t k_t^\top$

our state update rule is actually one step in gradient descent, trying to minimize a reconstruction!

Method	Online Learning Objective
LA	$\|\mathbf{S}_t - \mathbf{S}_{t-1}\|_F^2 - 2\langle \mathbf{S}_t \boldsymbol{k}_t, \boldsymbol{v}_t \rangle$
Mamba2	$\|\mathbf{S}_t - \alpha_t \mathbf{S}_{t-1}\|_F^2 - 2\langle \mathbf{S}_t \boldsymbol{k}_t, \boldsymbol{v}_t \rangle$
Longhorn	$\|\mathbf{S}_t - \mathbf{S}_{t-1}\|_F^2 - \beta_t \|\mathbf{S}_t \boldsymbol{k}_t - \boldsymbol{v}_t\|^2$
DeltaNet	$\|\mathbf{S}_t - \mathbf{S}_{t-1}\|_F^2 - 2\langle \mathbf{S}_t \boldsymbol{k}_t, \beta_t (\boldsymbol{v}_t - \mathbf{S}_{t-1} \boldsymbol{k}_t) \rangle$
Gated DeltaNet	$\|\mathbf{S}_t - \alpha_t \mathbf{S}_{t-1}\|_F^2 - 2\langle \mathbf{S}_t \boldsymbol{k}_t, \beta_t (\boldsymbol{v}_t - \alpha_t \mathbf{S}_{t-1} \boldsymbol{k}_t) \rangle$

Theorem 1. *Consider the TTT layer with $f(x) = Wx$ as the inner-loop model, batch gradient descent with $\eta = 1/2$ as the update rule, and $W_0 = 0$. Then, given the same input sequence $x_1, \ldots, x_T$, the output rule defined in Equation 5 produces the same output sequence $z_1, \ldots, z_T$ as linear attention.*

Proof. By definition of ℓ in Equation 4, $\nabla \ell(W_0; x_t) = -2(\theta_V x_t)(\theta_K x_t)^T$. By definition of batch GD in Equation 6 :

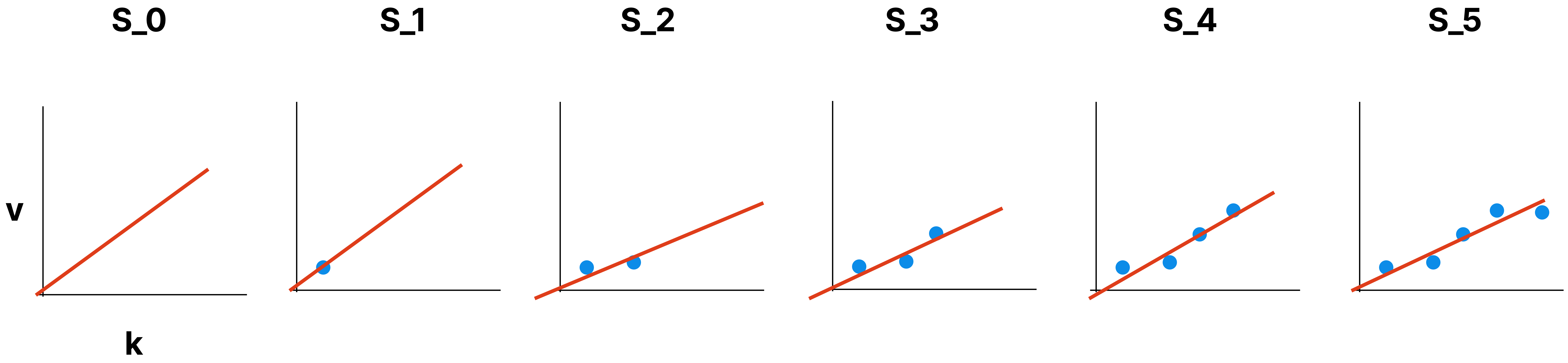
$$W_t = W_{t-1} - \eta \nabla \ell(W_0; x_t) = W_0 - \eta \sum_{s=1}^{t} \nabla \ell(W_0; x_s) = \sum_{s=1}^{t} (\theta_V x_s)(\theta_K x_s)^T.$$

Plugging W_t into the output rule in Equation 5, we obtain the output token:

$$z_t = f\left(\theta_Q x_t; W_t\right) = \sum_{s=1}^{t} (\theta_V x_s)(\theta_K x_s)^T (\theta_Q x_t),$$

which is the definition of linear attention. □

1-D Learning at Test Time



Learning at Test Time

https://arxiv.org/pdf/2407.04620

recall:

Given

1. **Running state (values accumulated with feature-mapped keys):**

$$S_i = \sum_{j=1}^{i} \phi(K_j) V_j^\top = S_{i-1} + \phi(K_i) V_i^\top \quad \in \mathbb{R}^{d_\phi \times d_v}$$

2. **Output definition (linear attention form):**

$$O_i = \frac{\phi(Q_i)^\top \sum_{j=1}^{i} \phi(K_j) V_j^\top}{\phi(Q_i)^\top \sum_{j=1}^{i} \phi(K_j)}$$

Substitute S_i

The numerator is exactly $\phi(Q_i)^\top S_i$.
The denominator is a scalar normalization term; define the running key-sum

$$Z_i = \sum_{j=1}^{i} \phi(K_j) \quad \in \mathbb{R}^{d_\phi}$$

Then:

$$O_i = \frac{\phi(Q_i)^\top S_i}{\phi(Q_i)^\top Z_i}$$

notice:

z_t = f(x_t; S_t)

S_t = S_t-1 - \eta \del_S l(S_t-1; x_t)

drawbacks

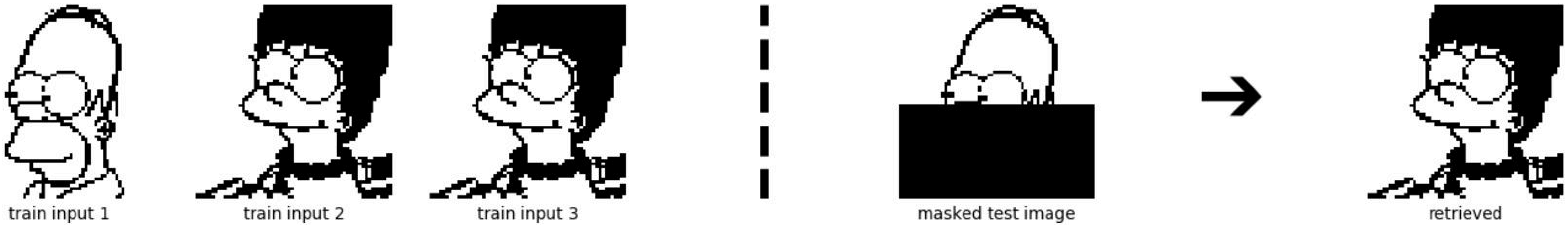
online learning requires "juicing the state" - ie you start with S_0 = 0

if you're juicing it with gradient updates, token by token, it might take a lot of tokens to regress effectively. Not sample efficient.

appendix

What do we mean by memory

"outer-product based associative memory"



how many memories can we store? ⟶ **what is the error rate on memory retrieval?**

Hopfield Network (binary, quadratic energy): .14N memories for N nodes.

Modern Hopfield Network (exponential energy, continuous): *exponential* memories for N nodes.

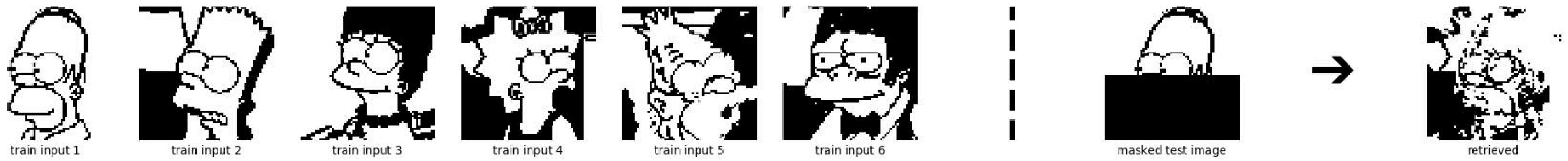
notably, one iteration of MHN = one self-attention pass.
Self-attention is a single Hopfield descent step on a query-conditioned exponential energy landscape.

https://arxiv.org/pdf/1702.01929

https://www.pnas.org/doi/epdf/10.1073/pnas.79.8.2554

Memory capacity

N nodes → .14N memories with low recall error, when the memories are minimally correlated.



How can we improve our memory capacity?

https://x.com/Grad62304977/status/1983900063917746224

https://x.com/rasbt/status/1984617030356451642

https://hanlab.mit.edu/blog/infinite-context-length-with-global-but-constant-attention-memory

https://srush.github.io/annotated-s4/