Test time regression

A unifying framework for designing sequence models with **associative memory**

Alex Wang @ Stanford University
ASAP Seminar | 2025-02-19

Paper link
Find me on X @heyyalexwang

Why are sequence models ubiquitous?

Sequence models as a universal abstraction

Sequences as a way to represent **information**:

Sequences as a way to represent **computation**:

Learning any transformation on any data format is then “just” sequence-to-sequence learning.

If everything can be reduced to sequences, then...

Maybe we just need to design better sequence models...

But how...?

Hint: add test-time associative memory

Overview of architectures unified by the test-time regression framework

Sequence models with test-time associative memory

Self-attention (Vaswani et al., 2017)

Linear regression layers:

- Mesa-layer (von Oswald et al., 2023)
- Intention (Garnelo et al., 2023)

Linear attention (Katharopoulos et al., 2020)

Feature-mapped linear attention:

- Performer (Choromanski et al., 2021)
- RFA (Peng et al., 2021)
- cosFormer (Qin et al., 2022)
- Hedgehog (Zhang et al., 2024)
- Based (Arora et al., 2024)
- ReBased (Arksenov et al., 2024)
- DiJiang (Chen et al., 2024)

Online learners/Fast-weight programmers:

- DeltaNet (Schlag et al., 2021)
- Longhorn (Liu et al., 2024)
- TTT layers (Sun et al., 2024)
- Gated DeltaNet (Yang et al., 2024)
- Titans (Behrouz et al., 2024)
- DeltaProduct (Siems et al., 2025)

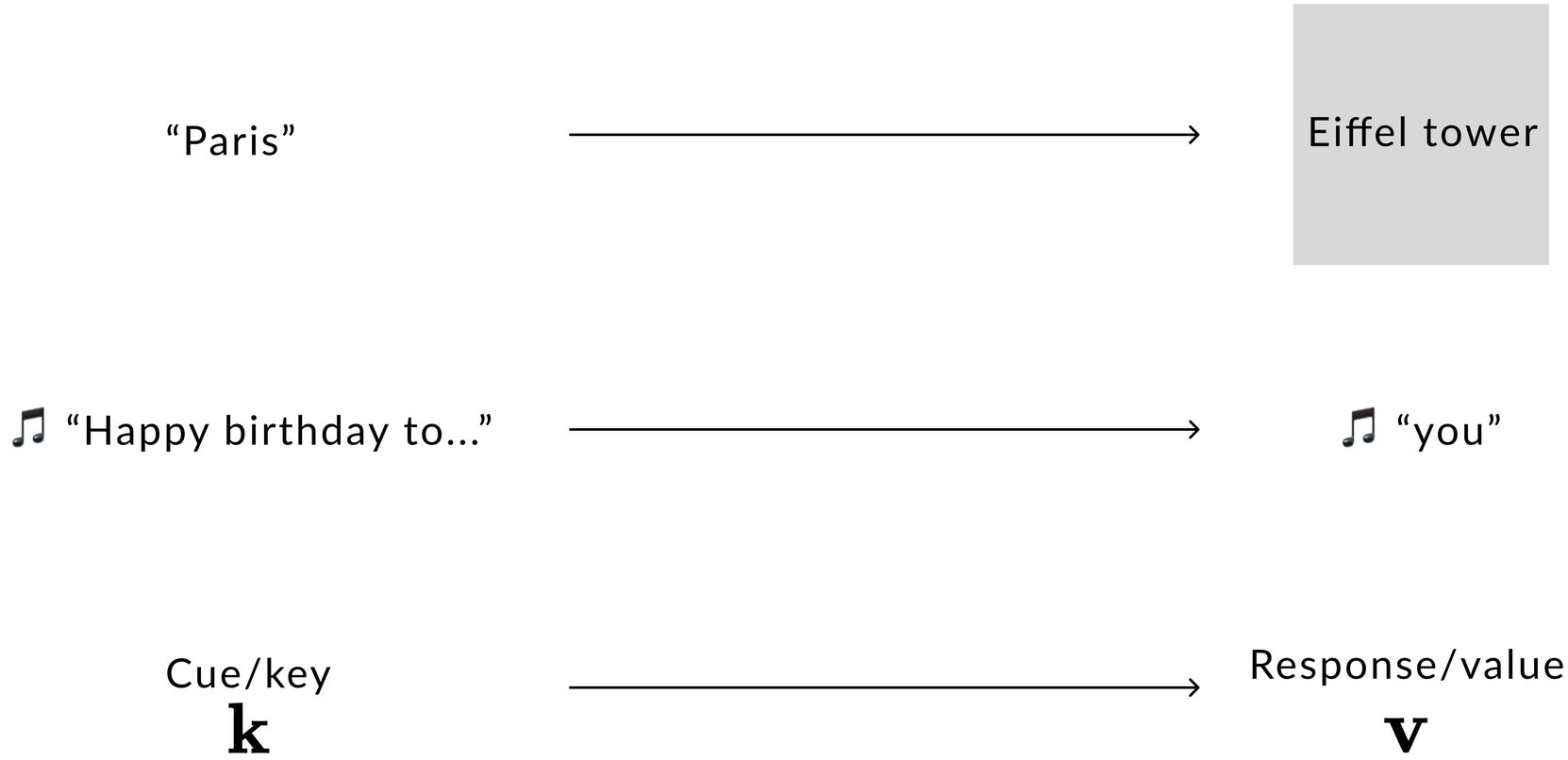
Gated variants of linear attention:

- Gateloop (Katsch 2023)
- HGRN-2 (Qin et al., 2023)
- LRU (Orvieto et al., 2023)
- Mamba (Gu et al., 2023, Dao et al., 2024)
- RWKV-6 (Peng et al., 2024)
- mLSTM (Beck et al., 2024)
- Gated Linear Attention (Yang et al., 2024)

What is associative memory?

Background on associative memory

> *associative memory is defined as the ability to learn and remember the relationship between unrelated items*



Why do we need associative memory?

Associative recall and in-context learning

Example from Arora et al., 2023:

Context:

“Hakuna Matata! It means no worries for the rest of your days! Hakuna Matata means no…”

Next-token:

“worries”

Intuition: a good next-token predictor should effectively use contextual clues at test-time

One way to do this is to have an associative “**working memory**” at test-time

Similar findings from Elhage et al., 2021, Olsson et al., 2022 on transformer induction heads

Illustrative example

A simple mathematical model for associative memory

Suppose we have key-value pairs $(\mathbf{k}_1, \mathbf{v}_1), (\mathbf{k}_2, \mathbf{v}_2), \ldots, (\mathbf{k}_T, \mathbf{v}_T)$ where the keys are *orthonormal* $\mathbf{k}_i^\top \mathbf{k}_j = \delta_{ij}$

We can store these relationships into an outer-product associative memory (Kohonen 1972):

$$\mathbf{M}_T = \sum_{t=1}^{T} \mathbf{v}_i \mathbf{k}_i^\top$$

Prompting the matrix-valued associative memory with $\mathbf{k}_j$, we retrieve $\mathbf{v}_j$ (perfect recall):

$$\mathbf{M}_T \mathbf{k}_j = \sum_{i=1}^{T} \mathbf{v}_i \mathbf{k}_i^\top \mathbf{k}_j = \mathbf{v}_j$$

Limitation: orthonormality requires key embedding dimension to scale with T

This kind of outer-product associative memory is also known as linear attention (Katharopoulos *et al.*, *2020)*

Generalizing our previous example

Associative memory as regression

Suppose we have key-value pairs $(\mathbf{k}_1, \mathbf{v}_1), (\mathbf{k}_2, \mathbf{v}_2), \ldots, (\mathbf{k}_T, \mathbf{v}_T)$ with relative importance $\gamma_1, \ldots, \gamma_T$

An associative memory m that stores these relationships should have $m(\mathbf{k}_i) \approx \mathbf{v}_i$ for all stored pairs

Hence memorizing key-value pairs into m reduces to solving a **regression** problem:

$$\text{minimize}_{m \in \mathcal{M}} \frac{1}{2} \sum_{i=1}^{T} \gamma_i \left\| \mathbf{v}_i - m(\mathbf{k}_i) \right\|_2^2$$

Nice properties:

1. Querying the memory with $\mathbf{q} = \mathbf{k}_i + \text{noise}$ will return a value close to $\mathbf{v}_i$ (fuzzy memory)
2. The better we solve this objective (e.g. better optimizer or more flexible functions), the better we memorize

How to design your own sequence model

A 3-step recipe for deriving a sequence layer

The regression objective

$$\text{minimize}_{m \in \mathcal{M}} \frac{1}{2} \sum_{i=1}^{T} \gamma_i \left\| \mathbf{v}_i - m(\mathbf{k}_i) \right\|_2^2$$

requires you to specify three “ingredients”:

1. the relative importance of each association $\gamma_1, \ldots, \gamma_T$
2. the function class $\mathcal{M}$
3. the minimization algorithm (e.g. gradient descent, Newton’s method, etc.)

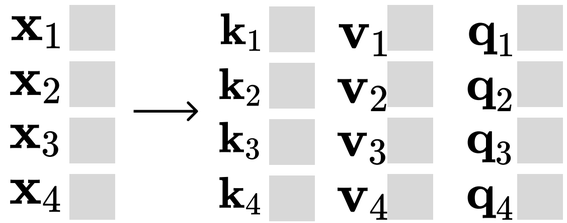
Once they are specified, just implement the minimization in the forward pass

Building an associative memory at test time

A general design pattern for sequence-to-sequence layers

Goal: transform input sequence to output sequence

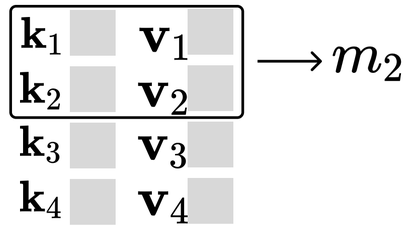
Preprocessing



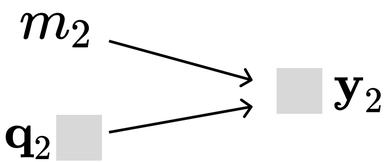
Transform inputs into queries, keys, and values

For $t = 1, \ldots, T$

$A_2 = \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{2}$



Select a subset of associations $A = \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i \in I}$ to memorize (e.g. prefix for causal tasks)



Query the memory to produce an output $\mathbf{y}_t = m_A(\mathbf{q}_t)$

Multiple set of keys/values/queries corresponds having multiple associative memory maps (similar to MHA)

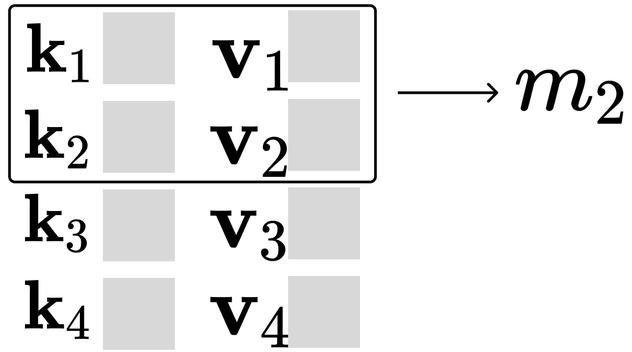
Sharing keys/values/queries corresponds to sharing regression inputs/outputs/prompts (similar to GQA)

For the rest of the talk, we will work with causal associative memory:

$$A = A_t := \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{t}$$

to memorize key-value pairs up to timestep t

$$A_2 = \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{2}$$

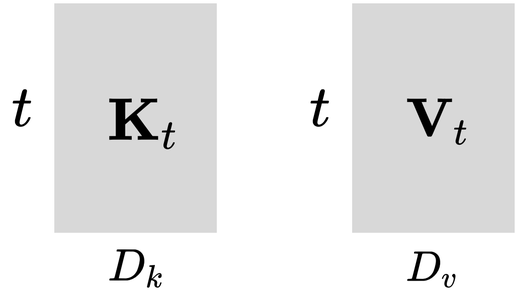


Vignette 1

Linear attention is suboptimal linear regression

Following our 3-step recipe, consider:

1. Choice of weighting: evenly weighted
2. Choice of function class: linear functions
3. Choice of optimizer: analytical solution



$$\mathbf{M}_t = \mathrm{argmin}_{\mathbf{M}} \frac{1}{2} \sum_{i=1}^{t} \|\mathbf{v}_i - \mathbf{M}\mathbf{k}_i\|_2^2 = \mathbf{V}_t^\top \mathbf{K}_t (\mathbf{K}_t^\top \mathbf{K}_t)^{-1}$$

$$\mathbf{y}_t = \mathbf{M}_t \mathbf{q}_t$$

This derives the mesa-layer (von Oswald et al., 2023), known to outperform standard linear attention

Dropping the inverse term, i.e. approximating the covariance with $\mathbf{K}_t^\top \mathbf{K}_t \approx \mathbf{I}$, derives linear attention:

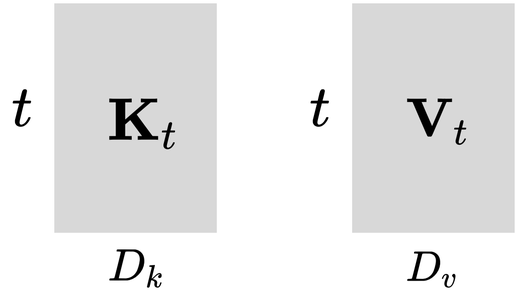
$$\mathbf{M}_t \approx \mathbf{V}_t^\top \mathbf{K}_t = \sum_{i=1}^{t} \mathbf{v}_i \mathbf{k}_i^\top = \mathbf{M}_{t-1} + \mathbf{v}_t \mathbf{k}_t^\top$$

Vignette 2

Gated linear attention is suboptimal weighted linear regression

For next-token prediction, recent timesteps are more important.

Hence, consider geometrically-decay weights instead: $\gamma_i^{(t)} = \gamma_{i+1}\gamma_{i+2}\cdots\gamma_t$



The analytical solution is then

$$\mathbf{M}_t = \mathrm{argmin}_{\mathbf{M}} \frac{1}{2}\sum_{i=1}^{t}\gamma_i^{(t)}\|\mathbf{v}_i - \mathbf{M}\mathbf{k}_i\|_2^2 = \mathbf{V}_t^\top\mathbf{\Gamma}_t\mathbf{K}_t(\mathbf{K}_t^\top\mathbf{\Gamma}_t\mathbf{K}_t)^{-1} \quad (\mathbf{\Gamma}_t)_{ii} = \gamma_i^{(t)}$$

$$\mathbf{y}_t = \mathbf{M}_t\mathbf{q}_t$$

Dropping the inverse term like before, we derive the update rule for linear attention with a forgetting gate:

$$\mathbf{M}_t \approx \mathbf{V}_t^\top\mathbf{\Gamma}_t\mathbf{K}_t\mathbf{q}_t = \sum_{i=1}^{t}\gamma_i^{(t)}\mathbf{v}_i\mathbf{k}_i^\top = \gamma_t\mathbf{M}_{t-1} + \mathbf{v}_t\mathbf{k}_t^\top$$

An alternative perspective

Linear attention implicitly computes batch gradient descent

Suppose instead we use gradient descent to minimize $\mathcal{L}_t(\mathbf{M}) = \frac{1}{2}\sum_{i=1}^{t} \|\mathbf{v}_i - \mathbf{M}\mathbf{k}_i\|_2^2$:

1. Initialize at the origin $\mathbf{M}_t^{(0)} = \mathbf{0}$
2. Perform one update: $\mathbf{M}_t^{(1)} = \mathbf{M}_t^{(0)} - \beta_1 \nabla \mathcal{L}_t(\mathbf{M}_t^{(0)}) = \mathbf{V}_t^\top \mathbf{K}_t$ when step size $\beta_t = 1$

since the gradient is $\nabla \mathcal{L}_t(\mathbf{M}) = (\mathbf{M}\mathbf{K}_t^\top - \mathbf{V}_t^\top)\mathbf{K}_t$

After only one gradient step, we arrive at the equations for linear attention

Thus linear attention, and its gated variant, is a **first-order method** for doing test-time regression!

In contrast, the analytical solution is equivalent to one step of Newton's method, a **second-order method.**

The difference comes from accounting for the curvature of the objective, governed by the covariance $\mathbf{K}_t^\top \mathbf{K}_t$

Vignette 3 Part 1

Fast-weight programmers as SGD

Having done full batch gradient descent, consider online/stochastic gradient descent instead:

$$\mathcal{L}_t(\mathbf{M}) = \frac{1}{2}\sum_{i=1}^{t} \|\mathbf{v}_i - \mathbf{M}\mathbf{k}_i\|_2^2 := \sum_{i=1}^{t} L_i(\mathbf{M})$$

At each step t, we use the latest $(\mathbf{k}_t, \mathbf{v}_t)$ to perform a *single-example* gradient step, using each iterate to store the associations in A_t:

$$\mathbf{M}_t = \mathbf{M}_{t-1} - \beta_t \nabla L_t(\mathbf{M}_{t-1}) = \mathbf{M}_{t-1}(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top) + \beta_t \mathbf{v}_t \mathbf{k}_t^\top$$

This derives the update rule of DeltaNet (Schlag et al., 2021), later parallelized by Yang et al., 2024.

Vignette 3 Part 2

Online learners perform SGD with adaptive step sizes

Liu et al. 2024 derived a recurrence for Longhorn via an online-learning objective,

$$\begin{aligned}\mathbf{M}_t &= \operatorname{argmin}_{\mathbf{M}} \frac{1}{2}\|\mathbf{M} - \mathbf{M}_{t-1}\|_F^2 + \frac{\delta_t}{2}\|\mathbf{v}_t - \mathbf{M}\mathbf{k}_t\|_2^2 \\ &= \mathbf{M}_{t-1}\left(\mathbf{I} - \frac{1}{1+\delta_t\|\mathbf{k}_t\|_2^2}\mathbf{k}_t\mathbf{k}_t^\top\right) + \frac{1}{1+\delta_t\|\mathbf{k}_t\|_2^2}\mathbf{v}_t\mathbf{k}_t^\top\end{aligned}$$

Comparing to our SGD update, we see that this is equivalent to using a data-adaptive step size

$$\beta_t = \frac{1}{1+\delta_t\|\mathbf{k}_t\|_2^2}$$

Vignette 3 Part 3

SGD with L2 regularization

Consider minimizing a regularized least squares objective $\mathcal{L}_{\text{reg},t}(\mathbf{M}) = \sum_{i=1}^{t} L_i(\mathbf{M}) + \frac{\lambda_i}{2}\|\mathbf{M}\|_2^2$

The update rule of SGD then becomes

$$\mathbf{M}_t = \mathbf{M}_{t-1}\left[(1-\beta_t\lambda_t)\mathbf{I} - \beta_t\mathbf{k}_t\mathbf{k}_t^\top\right] + \beta_t\mathbf{v}_t\mathbf{k}_t^\top$$

equivalent to Gated DeltaNet (Yang et al., 2024) after an invertible reparameterization, decoupling weight decay from the learning rate like AdamW (Loshchilov et al., 2019)

The function classes so far have all been linear $m_t(\mathbf{k}) = \mathbf{M}_t\mathbf{k}$ ...

Can we develop nonlinear associative memory?

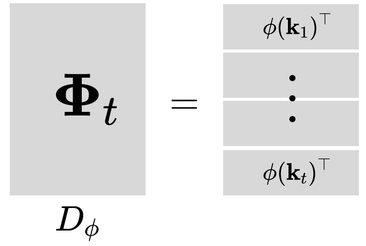
Yes!

But first... any questions so far?

The simplest nonlinear associative memory

Nonlinear associative memory by featurizing linear attention

Instead of using the original query-key space, we can transform them via a feature map ϕ



$$\mathbf{M}_t = \mathbf{argmin}_{\mathbf{M}} \frac{1}{2} \sum_{i=1}^{t} \|\mathbf{v}_i - \mathbf{M}\phi(\mathbf{k}_i)\|_2^2 = \mathbf{V}_t^\top \mathbf{\Phi}_t (\mathbf{\Phi}_t^\top \mathbf{\Phi}_t)^{-1}$$

$$\mathbf{y}_t = \mathbf{M}_t \phi(\mathbf{q}_t) = \sum_{i=1}^{t} \mathbf{v}_i \phi(\mathbf{k}_i)^\top \phi(\mathbf{q}_t)$$

Examples of feature maps include:

- 1 + ELU map (Katharopoulos et al., 2020)
- RELU (Kasai et al., 2021)
- Cosine map (Qin et al., 2021)
- Deterministic parameter-free projections (Schlag et al., 2021)
- SiLU (first used by Gu et al., 2023)
- Polynomial feature maps (Poggio 1975 ; Zhang et al. 2024; Arora et al., 2024)
- Random features: RFA (Peng et al., 2021), Performer (Choromanski et al., 2021), DiJiang (Chen et al., 2024)

Going infinite

Kernel regression with infinite-dimensional feature maps

To further increase the flexibility of our function class, we can apply the kernel trick: $\phi(\mathbf{k}_i)^\top \phi(\mathbf{k}_j) \to k(\mathbf{k}_i, \mathbf{k}_j)$

Querying our associative memory the produces the output

$$\mathbf{y}_t = m_t(\mathbf{q}_t) = \mathbf{V}_t^\top k(\mathbf{K}_t, \mathbf{K}_t)^{-1} k(\mathbf{K}_t, \mathbf{q}_t)$$

Kernel-regression-as-a-layer and its approximations has used in many past architectures, including SOFT (Lu et al., 2021), Performers (Choromanski et al., 2021), Skyformer (Chen et al., 2021), and intention (Garnelo et al., 2023)

When $k(\mathbf{k}_i, \mathbf{k}_j) = \exp(\mathbf{k}_i^\top \mathbf{k}_j / \sqrt{D_k})$, and we approximate the inverse with the identity, we derive an *unnormalized* softmax attention. Can we do better?

Background on a classic nonparametric regressor

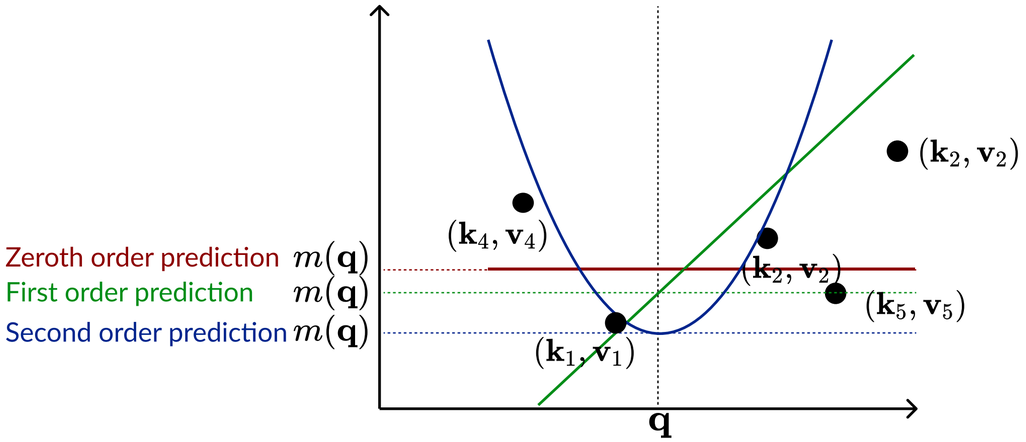
Intuition for local polynomial estimators

Suppose we have observations $(\mathbf{k}_1, \mathbf{v}_1), \ldots, (\mathbf{k}_t, \mathbf{v}_t)$.

We want to use all our observations to predict the value associated with a query $\mathbf{q}$.

What can we do?

1. Fit an order p polynomial centered at $\mathbf{q}$
2. Use the polynomial's value $m(\mathbf{q})$ at $\mathbf{q}$ as the prediction



Intuition: use observed data to find the best Taylor expansion around a new point $\mathbf{q}$

Back to architecture design

Local polynomial associative memory

Typically, local polynomial regression is done for 1D data. We generalize it to the multivariate case.

Following our 3-step recipe:

1. Choice of weights: weight each point by a monotonic function of the distance $\gamma_i = s(\mathbf{k}_i, \mathbf{q}) = s(\mathbf{k}_i - \mathbf{q})$, since we want to fit a *local* polynomial
2. Choice of function class: the set of order p polynomials around $\mathbf{q}$:

$$m(\mathbf{k}_i) = \mathbf{M}^{(0)} + \mathbf{M}^{(1)}(\mathbf{k}_i - \mathbf{q}) + \ldots + \mathbf{M}^{(p)}(\underbrace{\mathbf{k}_i - \mathbf{q}, \ldots, \mathbf{k}_i - \mathbf{q}}_{p \text{ arguments}})$$

3. Choice of optimizer: analytical solution, solving for $(\mathbf{M}^{(0)}, \ldots, \mathbf{M}^{(p)})$ where each $\mathbf{M}^{(j)}$ is an order j tensor/multilinear map

Deriving attention

Self-attention with QKNorm is a locally constant regressor

Solving the zeroth order (locally constant) estimator, we get self-attention:

$$\mathbf{y}_t = m_t(\mathbf{q}_t) = \operatorname{argmin}_{\mathbf{M}^{(0)}} \frac{1}{2}\sum_{i=1}^{t} s(\mathbf{k}_i, \mathbf{q}_t)\|\mathbf{v}_i - \mathbf{M}^{(0)}\|_2^2 = \sum_{i=1}^{t} \frac{s(\mathbf{k}_i, \mathbf{q}_t)}{\sum_{j=1}^{t} s(\mathbf{k}_j, \mathbf{q}_t)}\mathbf{v}_i$$

When keys and queries are normalized to unit length, the exponential smoothing kernel recovers the exponential dot product, and our weights are monotonic with respect to the distance

$$s(\mathbf{k}, \mathbf{q}) = \exp\left(\frac{-\|\mathbf{k}-\mathbf{q}\|_2^2}{2\sqrt{D_k}}\right) = \exp\left(\frac{-\|\mathbf{k}\|_2^2 - \|\mathbf{q}\|_2^2 + 2\mathbf{k}^\top\mathbf{q}}{2\sqrt{D_k}}\right) \propto \exp\left(\frac{\mathbf{k}^\top\mathbf{q}}{\sqrt{D_k}}\right)$$

Corollary: we can also derive higher order variants of self-attention!

A direct comparison

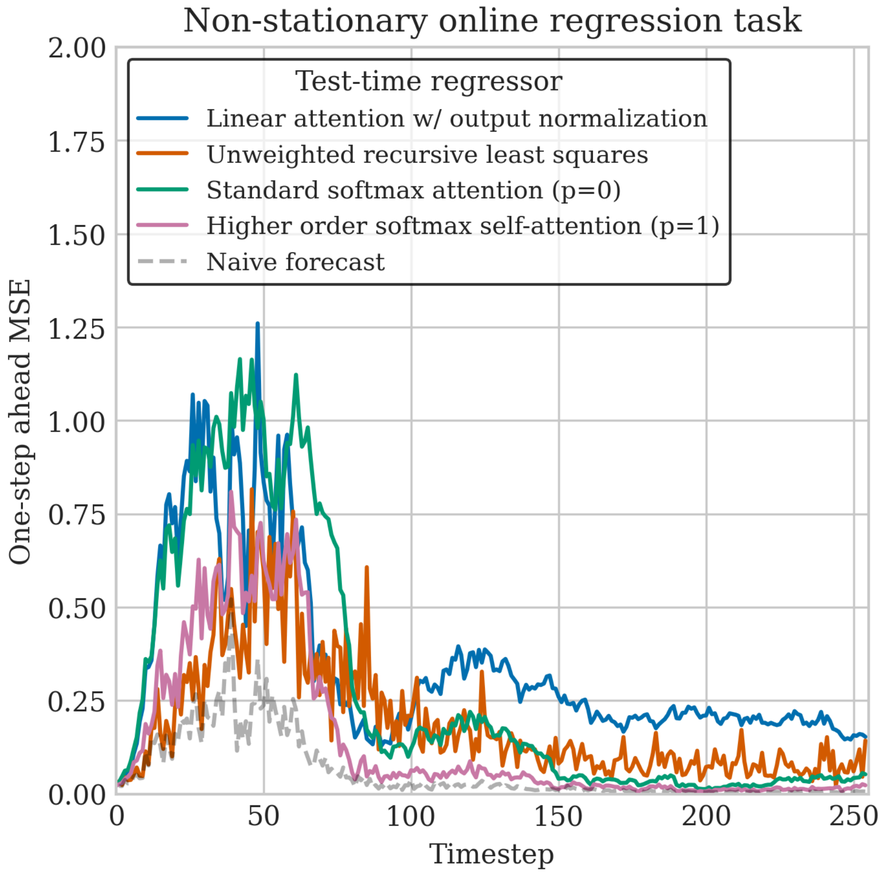
Solving regression with a single forward pass

We generate a set of keys and values from a non-stationary (switching) process, simulating a next-token prediction task:

$$\mathbf{k}_{t+1} = \begin{cases} 0.999\mathbf{k}_t + 0.02\boldsymbol{\varepsilon}_t & t < T/4 \\ 0.9\mathbf{k}_t + 0.02\boldsymbol{\varepsilon}_t & t \geq T/4 \end{cases}$$

$$\mathbf{v}_t = \mathbf{k}_{t+1}/\|\mathbf{k}_{t+1}\|_2 \qquad \boldsymbol{\varepsilon}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$$

Using the regression-memory correspondence, we apply each test-time regression layer to the key-value pairs *without* any learnable parameters



We've defined the test-time regression models, but what about the test-time regression **data**?

What key-value associations should we create?

For creating associations

Short convolution is all you need (on MQAR)

To predict “worries” given context

“Hakuna Matata! It means no worries for the rest of your days! Hakuna Matata means no…”

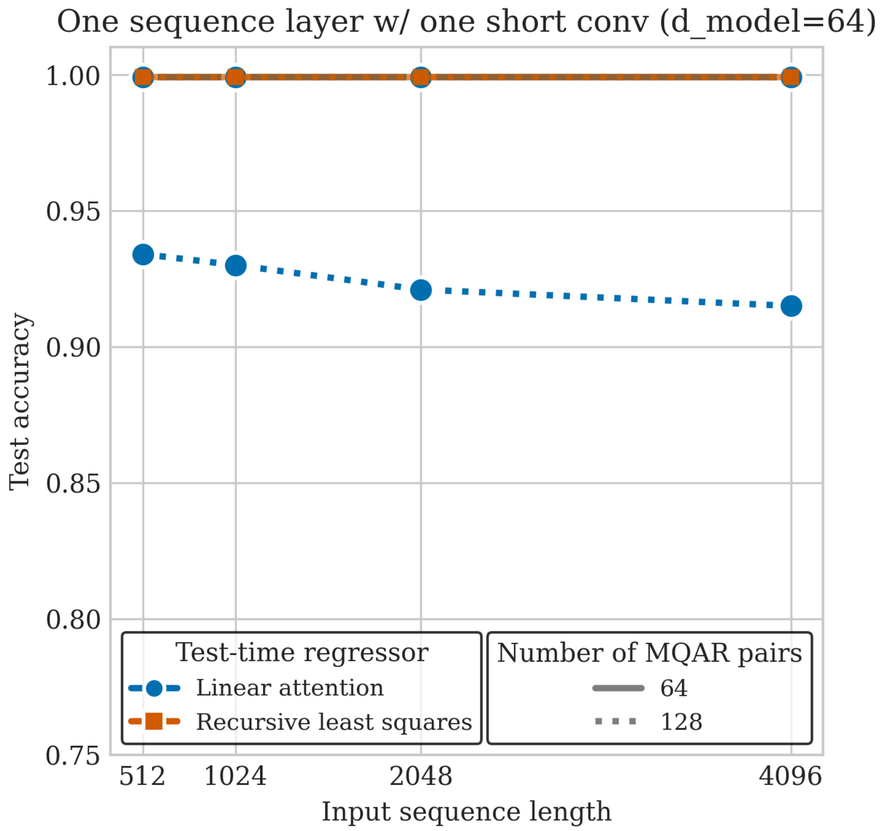
it suffices to use a short convolution to “look back”/shift:

$$\mathbf{k}_t = w_0\mathbf{x}_t + w_1\mathbf{x}_{t-1} = \mathbf{x}_{t-1}$$

$$\mathbf{v}_t = \mathbf{x}_t$$

The associative memory then stores adjacent pairs, such as (Hakuna, Matata) and (no, worries)

Short conv imitates an induction head of a transformer (first introduced in H3 by Fu et al., 2023)



Summary

A unified framework for sequence model design

Parametric regression (first order optimizer)	Parametric regression (second order optimizer)	Nonparametric regression
Batch gradient descent Linear attention, Mamba, GLA, HGRN, Gateloop, RWKV, RetNet, mLSTM, LRU	**Newton's method** *Mesa-layer*	**Kernel regression** *Intention*
Batch gradient descent with nonlinear feature maps Performer, cosFormer, RFA, Hedgehog, Based, Rebased, DiJiang		**Local polynomial estimation** *Self-attention* & *higher order generalizations*
Stochastic gradient descent DeltaNet, TTT, DeltaProduct Longhorn (adaptive step size) Gated DeltaNet (L2 regularization) Titans (momentum)		

All of these sequence layers construct and query an **associative memory via test-time regression** in their forward pass

Parametric associative memory usually has an efficient **recurrent update**, at the cost of forgetting the past

Speculations on future architectures...

💪 Overparameterized parametric regressors?

🪜 Parallelizable adaptive optimizers?

🧠

State-tracking? Episodic memory?

Life-long learning?

Other ways to use associative memory?

If you enjoyed this talk, you can find me on X @heyyalexwang!