Continual Learning & Memory Models (COMS6998)

Todd Morrill (tm3229@columbia.edu)

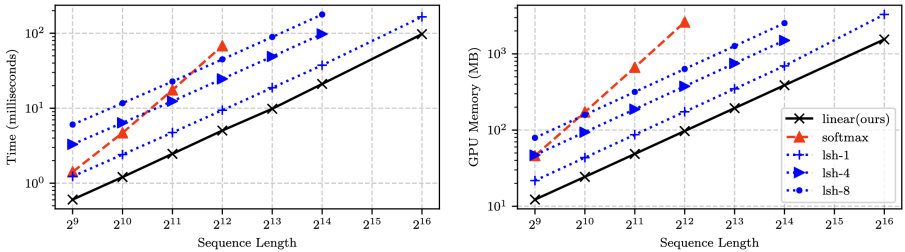
COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

February 2026

Outline

Transformers are RNNs

- Goal of the paper [Katharopoulos et al. 2020] is to tame the quadratic memory complexity ($O(T^2)$) and inference cost of transformers
- **Key idea** is to reformulate the self-attention layer as a recurrent neural network (RNN) with a fixed-size hidden state



Relevance for us

"This formulation is a first step towards better understanding the relationship between transformers and popular recurrent networks (Hochreiter & Schmidhuber, 1997) and the **processes used for storing and retrieving information.**"

Matrix Conventions

In this presentation, we use a left-multiplication convention for matrices and vectors because that's how it's laid out on hardware (e.g., tokens are rows in a matrix).

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_d \end{bmatrix} \in \mathbb{R}^{1\times d}, \quad \mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,d'} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,d'} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d,1} & w_{d,2} & \cdots & w_{d,d'} \end{bmatrix} \in \mathbb{R}^{d\times d'}$$

$$\mathbf{y} = \mathbf{x}\mathbf{W} = \begin{bmatrix} y_1 & y_2 & \cdots & y_{d'} \end{bmatrix} \in \mathbb{R}^{1\times d'}$$

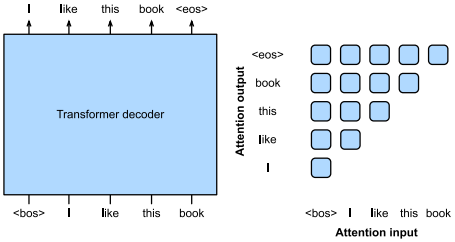
Transformer Recap

Recall that the self-attention layer computes the following

$$\mathbf{q}_t, \mathbf{k}_t, \mathbf{v}_t = \mathbf{x}_t \mathbf{W}_q, \mathbf{x}_t \mathbf{W}_k, \mathbf{x}_t \mathbf{W}_v \quad (1)$$

$$\mathbf{o}_t = \frac{\sum_{i=1}^{t} \exp(\mathbf{q}_t \mathbf{k}_i^\top) \mathbf{v}_i}{\sum_{j=1}^{t} \exp(\mathbf{q}_t \mathbf{k}_j^\top)} \quad (2)$$

where we have omitted the $\sqrt{d_k}$ scaling factor for clarity. Note the quadratic complexity in t.



Source:
https://d2l.ai/chapter_attention-mechanisms-and-transformers/large-pretraining-transformers.html

Linearizing Attention

What if we replace the exponential kernel $\exp(\mathbf{k}_i^\top \mathbf{q}_t)$ with the dot-product $\phi(\mathbf{k}_i)^\top \phi(\mathbf{q}_t)$ where $\phi : \mathbb{R}^d \to \mathbb{R}^n$ is a feature map (e.g., polynomial, ELU kernel, etc.)?

Linearizing Attention

What if we replace the exponential kernel $\exp(\mathbf{k}_i^\top \mathbf{q}_t)$ with the dot-product $\phi(\mathbf{k}_i)^\top \phi(\mathbf{q}_t)$ where $\phi : \mathbb{R}^d \to \mathbb{R}^n$ is a feature map (e.g., polynomial, ELU kernel, etc.)?

$$\mathbf{o}_t = \frac{\sum_{i=1}^{t} \phi(\mathbf{q}_t)\, \phi(\mathbf{k}_i)^\top \mathbf{v}_i}{\sum_{j=1}^{t} \phi(\mathbf{q}_t)\, \phi(\mathbf{k}_j)^\top} = \frac{\phi(\mathbf{q}_t)\left(\sum_{i=1}^{t} \phi(\mathbf{k}_i)^\top \mathbf{v}_i\right)}{\phi(\mathbf{q}_t)\left(\sum_{j=1}^{t} \phi(\mathbf{k}_j)^\top\right)} = \frac{\phi(\mathbf{q}_t)\mathbf{S}_t}{\phi(\mathbf{q}_t)\mathbf{z}_t^\top}, \tag{3}$$

$$\mathbf{S}_t = \sum_{i=1}^{t} \phi\big(\mathbf{k}_i\big)^\top \mathbf{v}_i \in \mathbb{R}^{n \times d}, \quad \mathbf{z}_t = \sum_{i=1}^{t} \phi\big(\mathbf{k}_i\big)^\top \in \mathbb{R}^n \tag{4}$$

How do we get $O(1)$ inference time?

There's still one problem: how do we compute $\mathbf{S}_t$ and $\mathbf{z}_t$ without performing a sum over all previous time steps? Naively, this still requires $O(T)$ computation at inference time.

How do we get $O(1)$ inference time?

There's still one problem: how do we compute $\mathbf{S}_t$ and $\mathbf{z}_t$ without performing a sum over all previous time steps? Naively, this still requires $O(T)$ computation at inference time.

$\mathbf{S}_t$ can be written recurrently as

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \phi(\mathbf{k}_t)^\top \mathbf{v}_t, \quad \mathbf{o}_t = \phi(\mathbf{q}_t)\mathbf{S}_t \tag{5}$$

where $\phi(\mathbf{q}_t)\,\mathbf{z}_t^\top$ is commonly dropped from the computation for numerical stability.

Takeaways

- The "working memory" of the linear Transformer is now only a fixed sized memory, as opposed to the original Transformer, which allows memory (i.e., keys and values) to grow unbounded.
- By linearizing the attention mechanism, we can reduce the memory cost from $O(T^2)$ to simply the size of the hidden state $\mathbf{S}_t$.
- Inference only requires updating $\mathbf{S}_t$ with the new key-value pair, resulting in $O(1)$ time complexity per output token.
- This formulation allows us to view transformers as RNNs with a fixed-size hidden state $\mathbf{S}_t$.

Takeaways

- The "working memory" of the linear Transformer is now only a fixed sized memory, as opposed to the original Transformer, which allows memory (i.e., keys and values) to grow unbounded.
- By linearizing the attention mechanism, we can reduce the memory cost from $O(T^2)$ to simply the size of the hidden state $\mathbf{S}_t$.
- Inference only requires updating $\mathbf{S}_t$ with the new key-value pair, resulting in $O(1)$ time complexity per output token.
- This formulation allows us to view transformers as RNNs with a fixed-size hidden state $\mathbf{S}_t$.

Takeaways

- The "working memory" of the linear Transformer is now only a fixed sized memory, as opposed to the original Transformer, which allows memory (i.e., keys and values) to grow unbounded.
- By linearizing the attention mechanism, we can reduce the memory cost from $O(T^2)$ to simply the size of the hidden state $\mathbf{S}_t$.
- Inference only requires updating $\mathbf{S}_t$ with the new key-value pair, resulting in $O(1)$ time complexity per output token.
- This formulation allows us to view transformers as RNNs with a fixed-size hidden state $\mathbf{S}_t$.

Takeaways

- The "working memory" of the linear Transformer is now only a fixed sized memory, as opposed to the original Transformer, which allows memory (i.e., keys and values) to grow unbounded.
- By linearizing the attention mechanism, we can reduce the memory cost from $O(T^2)$ to simply the size of the hidden state $\mathbf{S}_t$.
- Inference only requires updating $\mathbf{S}_t$ with the new key-value pair, resulting in $O(1)$ time complexity per output token.
- This formulation allows us to view transformers as RNNs with a fixed-size hidden state $\mathbf{S}_t$.

Capacity Limits of Outer Product Memories

How does the state matrix $\mathbf{S}_t = \sum_{i=1}^{t} \mathbf{k}_i \otimes \mathbf{v}_i$ behave as more key-value pairs are added?

Capacity Limits of Outer Product Memories

How does the state matrix $\mathbf{S}_t = \sum_{i=1}^{t} \mathbf{k}_i \otimes \mathbf{v}_i$ behave as more key-value pairs are added?

Each outer product increases the rank of $\mathbf{S}_t$ by at most 1, so after L steps, $\text{rank}(\mathbf{S}_L) \leq \min(L, d_k, d_v)$.

Capacity Limits of Outer Product Memories

How does the state matrix $\mathbf{S}_t = \sum_{i=1}^{t} \mathbf{k}_i \otimes \mathbf{v}_i$ behave as more key-value pairs are added?

Each outer product increases the rank of $\mathbf{S}_t$ by at most 1, so after L steps, $\text{rank}(\mathbf{S}_L) \leq \min(L, d_k, d_v)$.

When $L > d_k$ or d_v, new key-value pairs will interfere with previously stored pairs, leading to retrieval errors.

Capacity Limits of Outer Product Memories

How does the state matrix $\mathbf{S}_t = \sum_{i=1}^{t} \mathbf{k}_i \otimes \mathbf{v}_i$ behave as more key-value pairs are added?

Each outer product increases the rank of $\mathbf{S}_t$ by at most 1, so after L steps, $\text{rank}(\mathbf{S}_L) \leq \min(L, d_k, d_v)$.

When $L > d_k$ or d_v, new key-value pairs will interfere with previously stored pairs, leading to retrieval errors.

This is analogous to the capacity limits observed in Hopfield networks, where the number of stored patterns is limited by the number of neurons.

Capacity Limits of Outer Product Memories

How does the state matrix $\mathbf{S}_t = \sum_{i=1}^{t} \mathbf{k}_i \otimes \mathbf{v}_i$ behave as more key-value pairs are added?

Each outer product increases the rank of $\mathbf{S}_t$ by at most 1, so after L steps, $\text{rank}(\mathbf{S}_L) \leq \min(L, d_k, d_v)$.

When $L > d_k$ or d_v, new key-value pairs will interfere with previously stored pairs, leading to retrieval errors.

This is analogous to the capacity limits observed in Hopfield networks, where the number of stored patterns is limited by the number of neurons.

To mitigate interference, strategies such as orthogonalizing keys or implementing forgetting mechanisms can be employed.

Side Note: Outer Products Link Many Concepts

- The linear Transformer formalism led to DeltaNet [Schlag, Irie, and Schmidhuber 2021], which links to the Fast Weight Programmers (FWP) of the 1990s ($\mathbf{S}_t$ can be thought of as fast weights and model weights can be thought of as slow weights).
- The matrix-valued state is composed of outer-products between keys and values (i.e., $\mathbf{k} \otimes \mathbf{v}$), which brings in links to Tensor Product Representation Theory (see Smolensky), Hyperdimensional Computing (see Kanerva), Vector-Symbolic Architectures (see Plate), and Hopfield networks (see Hopfield, Krotov).
- Outer products allow for natural retrieval: $\mathbf{q}(\mathbf{k}^\top \mathbf{v}) = (\mathbf{q}\mathbf{k}^\top)\mathbf{v}$.

Updating Memories

DeltaNet [Schlag, Irie, and Schmidhuber 2021] addresses this capacity limit with a new update rule for $\mathbf{S}_t$.

Updating Memories

DeltaNet [Schlag, Irie, and Schmidhuber 2021] addresses this capacity limit with a new update rule for $\mathbf{S}_t$.

Potentially overwrite existing memories instead of just adding new ones:

$$\bar{\mathbf{v}}_t = \phi(\mathbf{k}_t)\mathbf{S}_{t-1} \qquad \text{(Existing key-value)}$$

$$\beta_t = \sigma(\mathbf{x}_t\mathbf{W}_\beta) \qquad \text{(Dynamic "learning rate")}$$

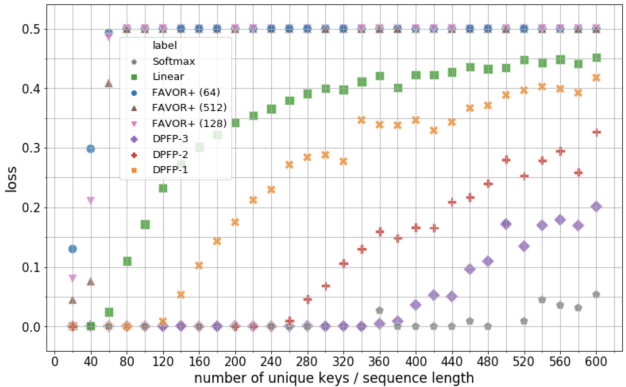
$$\mathbf{v}_{\text{new}} = \beta_t\mathbf{v}_t + (1-\beta_t)\bar{\mathbf{v}}_t \qquad \text{(Interpolate)}$$

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \underbrace{\phi(\mathbf{k}_t)^\top\mathbf{v}_{\text{new}}}_{\textit{Write}} - \underbrace{\phi(\mathbf{k}_t)^\top\bar{\mathbf{v}}_t}_{\textit{Remove}} \qquad \text{(Update memory)}$$

Different Kernel Functions

[Schlag, Irie, and Schmidhuber 2021] also examine how the kernel function, and in particular, its dimensionality, affects memory capacity and performance.

They evaluate them on needle-in-a-haystack retrieval tasks. See paper for more details.



Beyond Linear Updates: Memory as Optimization

- **Delta Rule Revisited:** DeltaNet [Schlag, Irie, and Schmidhuber 2021] interprets memory as an associative map $\mathbf{S}$ updated via an error-correction rule:

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \beta \underbrace{\mathbf{k}_t^\top (\mathbf{k}_t \mathbf{S}_{t-1} - \mathbf{v}_t)}_{\text{Gradient of what objective?}}$$

- **The Optimization View:** This update is exactly one step of gradient descent on the objective $\mathcal{L}_t(\mathbf{S}) = \frac{1}{2}\|\mathbf{k}_t\mathbf{S} - \mathbf{v}_t\|^2$.
- **Generalization (TTT):** Why limit memory to a linear map $\mathbf{S}$? [Sun et al. 2024]

Test Time Training (TTT)

Let the hidden state be the weights $\boldsymbol{\theta}$ of a neural network $f_{\boldsymbol{\theta}}$. The "forward pass" includes an online optimization of the objective:

$$\mathcal{L}_t(\boldsymbol{\theta}) = \frac{1}{2}\|f_{\boldsymbol{\theta}}(\mathbf{k}_t) - \mathbf{v}_t\|^2$$

Where this encourages the model to remember memories (i.e., $\mathbf{v}_t$).

The TTT Mechanism

- **The Hidden State is a Model:** In TTT [Sun et al. 2024], $\mathbf{S}_t$ is not a vector or a fixed matrix, but the parameters of a small sub-network (e.g., a linear layer or a small MLP).
- **Self-Supervised Memory:** At each step t, the model performs a "Test Time Training" step to update its state to account for the current input $\mathbf{x}_t$.
- This is a big unlock because now we can start thinking about what makes for a good memory subnetwork!

The State Space Lineage

- **The Bottleneck:** Transformer self-attention requires $O(T^2)$ time and memory to relate every token to every other token.
- **The Goal:** A model that compresses context into a **fixed-size latent state $\mathbf{h}_t$**, enabling $O(T)$ training and $O(1)$ inference.
- **State Space Models (SSMs):** Treat the sequence as a discretization of a continuous-time dynamical system [Gu and Dao 2024]:

Idealized Continuous-Time System

We model a 1D signal $x(t) \mapsto y(t)$ via an N-dimensional latent state $\mathbf{h}(t)$:

$$\mathbf{h}'(t) = \mathbf{h}(t)\mathbf{A} + x(t)\mathbf{B} \quad \text{(State Evolution)}$$
$$y(t) = \mathbf{h}(t)\mathbf{C} \quad \text{(Observation)}$$

where $\mathbf{A} \in \mathbb{R}^{N \times N}$, $\mathbf{B} \in \mathbb{R}^{1 \times N}$, and $\mathbf{C} \in \mathbb{R}^{N \times 1}$.

Discretization and Selection (Mamba)

To process discrete tokens, we discretize the system with step size Δ:
Zero-Order Hold (ZOH):

- $\overline{\mathbf{A}} = \exp(\Delta \mathbf{A})$
- $\overline{\mathbf{B}} = (\Delta \mathbf{A})^{-1}(\exp(\Delta \mathbf{A}) - \mathbf{I})\Delta \mathbf{B}$

Selective Recurrence (S6)

Mamba makes parameters $(\Delta, \mathbf{B}, \mathbf{C})$ **input-dependent**, allowing the model to "select" what to forget or remember:

$$\mathbf{h}_t = \mathbf{h}_{t-1}\overline{\mathbf{A}}_t + x_t\overline{\mathbf{B}}_t, \quad y_t = \mathbf{h}_t\mathbf{C}_t$$

Note: Unlike linear time invariant (LTI) models, Mamba's time-varying nature requires a hardware-aware parallel scan to maintain efficiency.

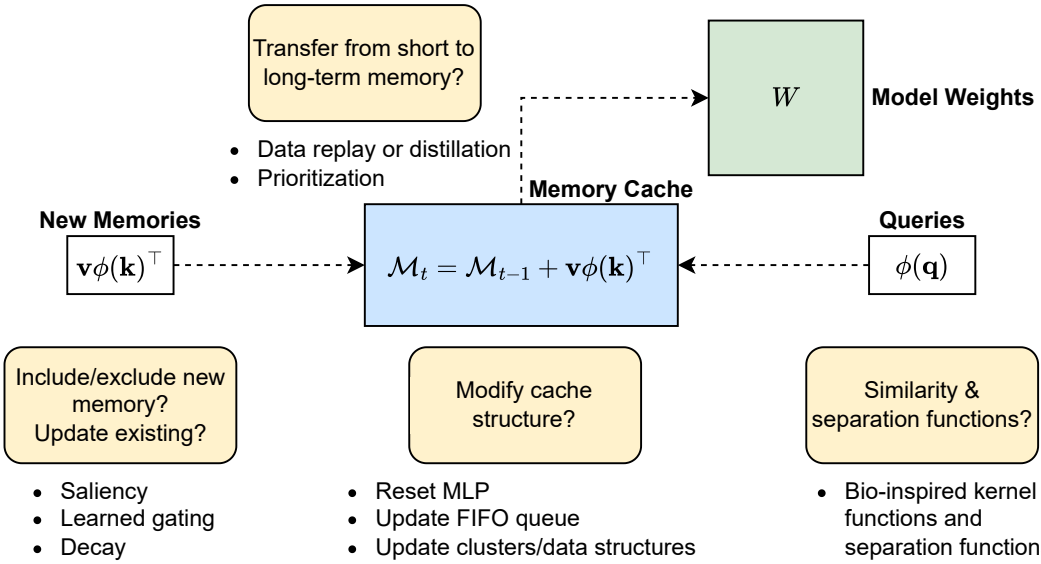
Conclusions

The Modern Recurrent Hierarchy

Method	**State Representation**	**Update Logic**
Mamba (S6)	Vector $\mathbf{h} \in \mathbb{R}^{N \times D}$	Selective Scalar Gating
DeltaNet	Matrix $\mathbf{S} \in \mathbb{R}^{d_k \times d_v}$	Linear Least Squares Step
TTT	Weights $\boldsymbol{\theta} \in \mathbb{R}^{P}$	Gradient Descent on $\mathcal{L}_t$

Conclusion: We have moved from "Look-back" (Attention) to "Dynamic Compression" (DeltaNet/SSMs) to "Online Learning" (TTT).

Some Research Ideas



Appendix

References I

- Gu, Albert and Tri Dao (May 2024). *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*. DOI: 10.48550/arXiv.2312.00752. arXiv: 2312.00752 [cs]. (Visited on 02/02/2026).
- Katharopoulos, Angelos et al. (Aug. 31, 2020). *Transformers Are RNNs: Fast Autoregressive Transformers with Linear Attention*. DOI: 10.48550/arXiv.2006.16236. arXiv: 2006.16236 [cs]. URL: http://arxiv.org/abs/2006.16236 (visited on 02/17/2025). Pre-published.
- Schlag, Imanol, Kazuki Irie, and Jürgen Schmidhuber (June 9, 2021). *Linear Transformers Are Secretly Fast Weight Programmers*. DOI: 10.48550/arXiv.2102.11174. arXiv: 2102.11174 [cs]. URL: http://arxiv.org/abs/2102.11174 (visited on 02/28/2025). Pre-published.
- Sun, Yu et al. (Aug. 11, 2024). *Learning to (Learn at Test Time): RNNs with Expressive Hidden States*. DOI: 10.48550/arXiv.2407.04620. arXiv: 2407.04620 [cs]. URL: http://arxiv.org/abs/2407.04620 (visited on 02/28/2025). Pre-published.