Linear Transformers Are Secretly Fast Weight Programmers

Imanol Schlag, Kazuki Irie, J¨urgen Schmidhuber

Presented By: Litong Liu and Giselle McPhilliamy

Georgia Tech

Introduction

Georgia Tech

The Transformer Dominance

- Transformers have become the dominant architecture for processing sequences:
  - Machine translation
  - Language modeling
  - Question answering
  - …
- Secrete of success: ***self-attention***

Self-Attention: The Core of Transformers

- Each token creates three vectors:

Query (q)	*"What am I looking for?"*
Key (k)	*"What do I contain?"*
Value (v)	*"What information do I offer?"*

- Output = weighted average of values (weighted by query-key similarity)

Standard Self-Attention

$$y^{(i)} = V^{(i)}\ \mathrm{softmax}\left(K^{(i)\intercal}\ q^{(i)} \right)$$

Every token compares against every other token

⬇️

$O(L^2)$ computation

Georgia Tech

Problem: The Cost of Quadratic Scaling

The quadratic cost becomes prohibitive as sequences grow:

Sequence Length	Comparisons (L^2)	KV Storage (L)
100 tokens	**10,000**	100 vectors
1,000 tokens	**1,000,000**	1,000 vectors
10,000 tokens	**100,000,000**	10,000 vectors

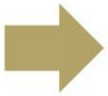
Consequence: Context window must be capped at fixed size! The model cannot see long range dependencies beyond this fixed window.

Linear Transformers: Trading Softmax for Efficiency

Standard Transformer

- *Compare query v.s. all keys individually*
- *Apply softmax for Attention weights*
- *Blend values by weights*

$O(L^2)$ time, O(L) memory, growing KV store

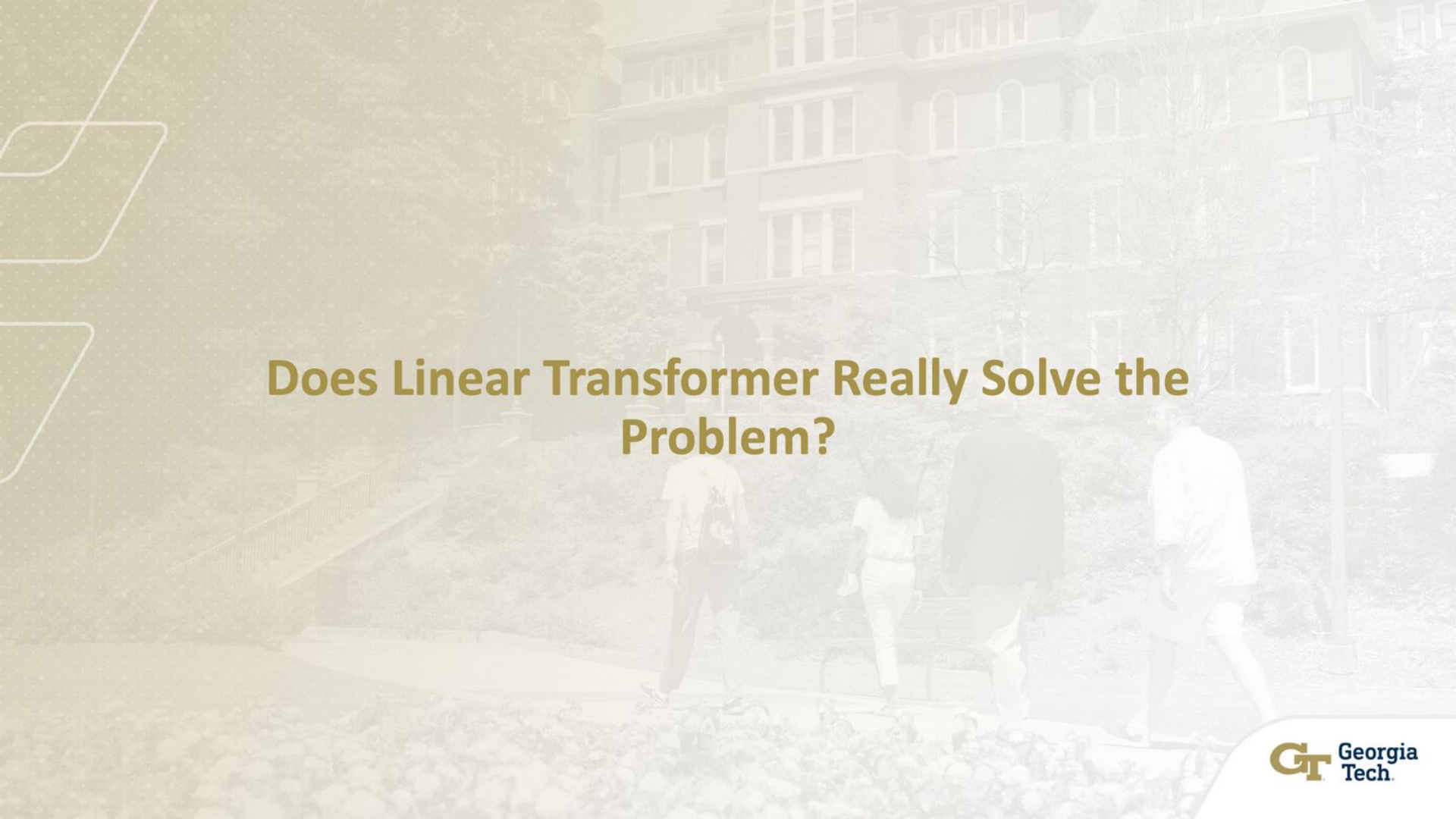


Linear Transformer

- *Replace softmax with kernel φ*
- *Attention score:* $\phi(k)\cdot\phi(q)$
- *Compress all history into one fixed size matrix*
- *Query the matrix directly to retrieve the output*

O(L) time, O(1) memory, fixed size matrix

Does Linear Transformer Really Solve the Problem?



But What Does Linear Attention Sacrifice?

- Linear Transformers solve the efficiency problem, but introduce new problems:

Bounded Memory Capacity

Fixed size matrix can only store a limited number of clean associations before interference

No Ability to Update

Purely additive writes – old associations can never be erased or corrected

These problems were not visible from the perspective of "approximating softmax" — a different lens was needed.

Prior Work and Key Insights

Prior Work

Associative Memory

Hebb (1949) , Steinbuch (1961) , Kohonen (1972), Hopfield (1982):

- Outer products store pre-defined patterns;
- Built networks based on outer product storage.

However, all these systems used **pre-defined patterns** — humans decided in advance what to store and retrieve.

Fast Weight Programmers

Schmidhuber:

- Introduced a crucial innovation: a slow network learns to generate keys and values via gradient descent.
- Combine through outer products and written into a fast weight matrix serving as dynamic memory.

For the first time, the patterns being stored were self-invented by the network, not hand-designed.

Linear Transformers

Katharopoulos, Choromanski, Peng:

- Linearize the softmax in self-attention for O(L) efficiency.

How Does Fast Weight Programmers Work

FWP has two operations on a fixed-size matrix W:

Write:

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

- The slow network reads input $x^{(i)}$ and produces a key k and value v through learned projections. Their outer product is added to the fast weight matrix. The outer product binds the key to the value — it's a pairing operation that encodes "this value goes with this key." The matrix accumulates all such associations as a superposition.

Read:

$$y^{(i)} = W^{(i)} q^{(i)}$$

- To retrieve, multiply the matrix by a query vector. If the query matches a stored key, the corresponding value is recovered through the dot product. This works cleanly when the stored keys are orthogonal to each other — their dot products are zero, so they don't interfere.

Georgia Tech

Problem of FWP

The matrix is fixed-size regardless of how many associations have been stored, giving constant memory:

the model can only add — it cannot erase or correct.

Key Insight: Linear Transformers = Fast Weight Programmers

With softmax:

$$y = V \cdot \mathrm{softmax}(K^T q) \quad \text{– nonlinear, cannot regroup} \rightarrow O(L^2)$$

Remove / linearize softmax

Without softmax:

$$y = (V \cdot K^T) \cdot q = [\Sigma\, v^{(j)} \otimes k^{(j)}] \cdot q = W \cdot q$$

This is exactly the FWP: accumulate outer products into W, read with query q

Georgia Tech

Key Insight: Linear Transformers = Fast Weight Programmers

With softmax:

$y = V \cdot \text{softmax}(K^T q)$ — nonlinear, cannot regroup → $O(L^2)$

Remove softmax

Without softmax:

$y = (V \cdot K^T) \cdot q = [\Sigma\, v^{(j)} \otimes k^{(j)}] \cdot q = W \cdot q$

This is exactly the FWP: accumulate outer products into W, read with query q

Replace softmax with decomposable kernel

LT: Repalce softmax: $k(k, q) = \exp(k \cdot q) = \varphi(k) \cdot \varphi(q)$

$W^{(i)} = W^{(i-1)} + v^{(i)} \otimes \phi(k^{(i)})$ — write (accumulate outer products)

$y^{(i)} = W^{(i)}\, \phi(q^{(i)}) \,/\, ((\Sigma\, \phi(k^{(j)}) \cdot \phi(q^{(i)}))$ — read (with normalization)

Why Does This Equivalence Matter?

- **Reveals capacity limits:**

  Associative memory theory tells us: at most d_dot orthogonal keys can be stored without interference

- **Reveals update weakness:**

  Additive outer products can never erase — once written, associations persist and interfere

- **Connects to known solutions:**

  Decades of research on error-correcting rules (the delta rule, 1960) and memory models become directly applicable

Technical Challenges

Challenge 1: Bounded Memory Capacity

Keys must be orthogonal for clean retrieval

- In d_dot dimensions → at most d_dot orthogonal keys
- Key #(d_dot + 1) must overlap with existing keys
- Overlap causes crosstalk — retrieval returns mixtures
- Typical d_dot = 64, but sequences can be 1000s of tokens

Analogy

64 hooks on a wall

64 bags → each has its own hook ✓

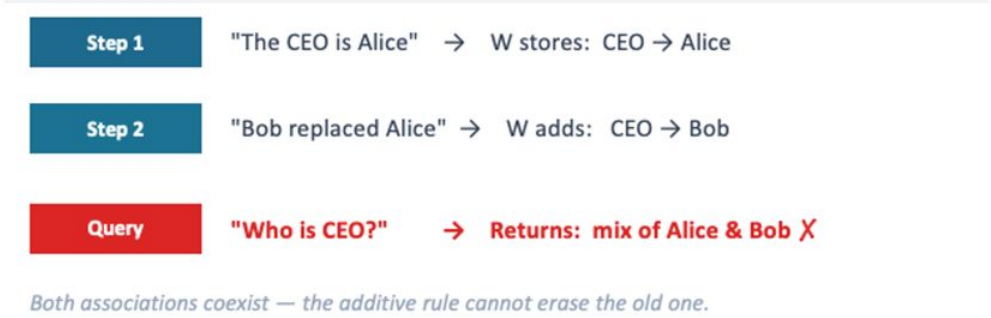
65th bag → must share a hook ✗
Bags get tangled on retrieval

How can we increase capacity without losing linear-time efficiency?

Georgia Tech

Challenge 2: No Ability to Update or Correct Stored Associations

The standard write rule simply adds the new outer product to the matrix. Every association ever written remains permanently.



How can we enable the model to correct outdated associations without disrupting unrelated ones already stored in memory?

Georgia Tech

Challenge 3: Designing a Better Kernel Function ϕ

Existing kernel functions are not good enough…

- **ELU+1:**

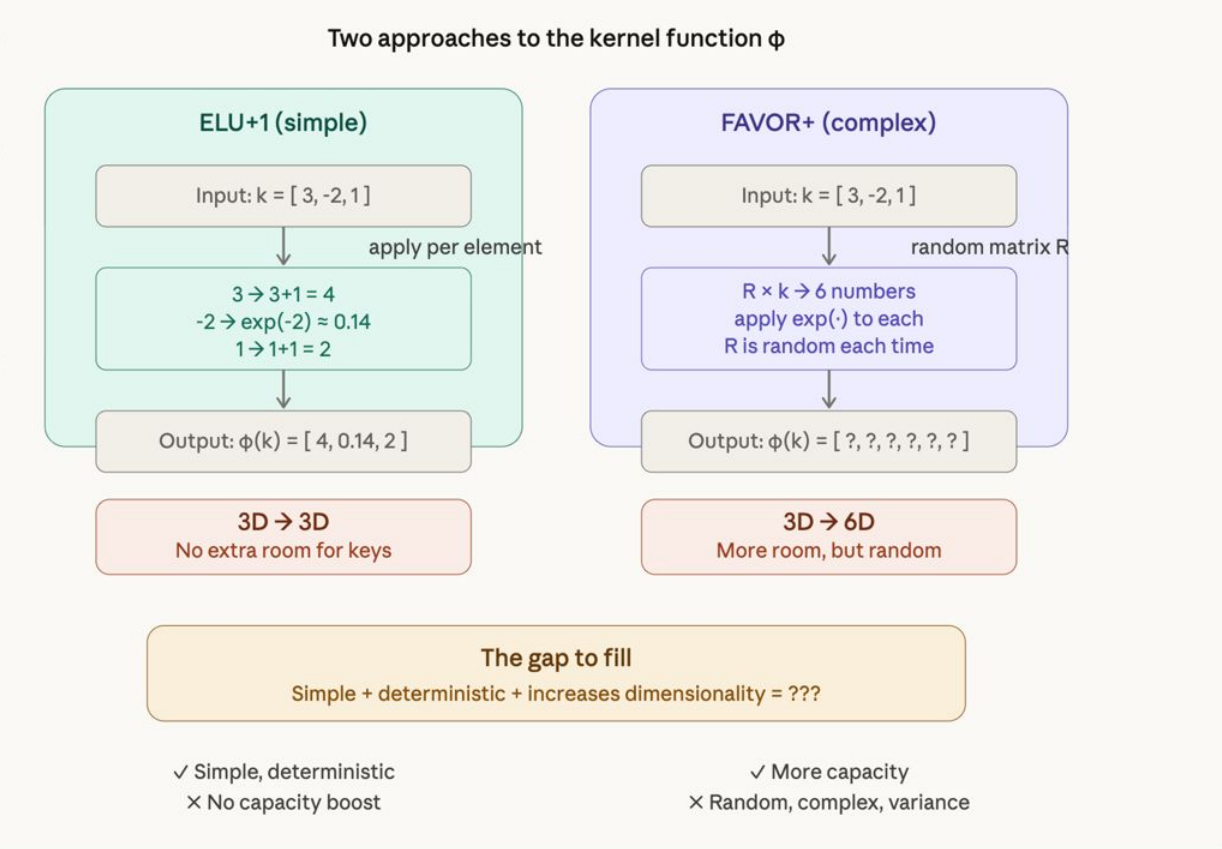
  Simple, fast, but d_dot = d_key: no capacity boost at all

- **FAVOR+:**

  Mathematically rigorous softmax approximation using random project. But introduce variance, add complexity, and never exact

> Can we design a kernel function that is simple and deterministic like ELU+1, while increasing the memory capacity like FAVOR+?

Challenge 3: Designing a Better Kernel Function ϕ



Can we design a kernel function that is simple and deterministic like ELU+1, while increasing the memory capacity like FAVOR+?

Georgia Tech

Solutions, Experiments, and Critical Analysis

Beyond Equivalence: The FWP Upgrade

Standard Linear Transformer

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

- Stores memory as sum of outer products
- Fixed-size memory → capacity limit
- Purely additive updates (no correction)

Delta Network

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

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

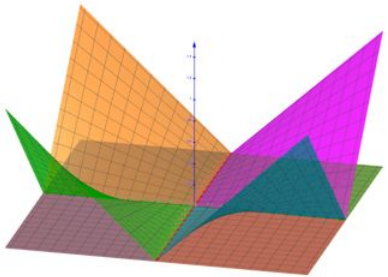
- Uses error-correcting update (delta rule)
- Can overwrite outdated information
- Improves memory interaction

- Improving Memory Capacity: **DPFP**
- Maintaining Stability: **Sum Normalization**
- Improving Interaction: **The Delta Rule**

Georgia Tech

DPFP: Fixing Memory Capacity

- Goal: Reduce interference between stored key-value pairs
- Method: Project keys into higher-dimensional space ($d_{key} \rightarrow d_{dot}$ where $d_{dot} > d\_key$)
- Effect: Makes keys more **orthogonal** → **cleaner retrieval**



More orthogonality = higher memory capacity

Figure 1. A visualisation of a DPFP from a 2d space (the xy-plane) to a 4d space (the four colored surfaces). Each surface is a partial function which represents one element of the 4d vector.

Georgia Tech

Visualizing the Projection Space

Each input activates a different region → prevents overlap

- Partial function: $\phi_i(k)$
- Enforces non-overlapping domains
- $d_{dot} = 2d_{key}v$

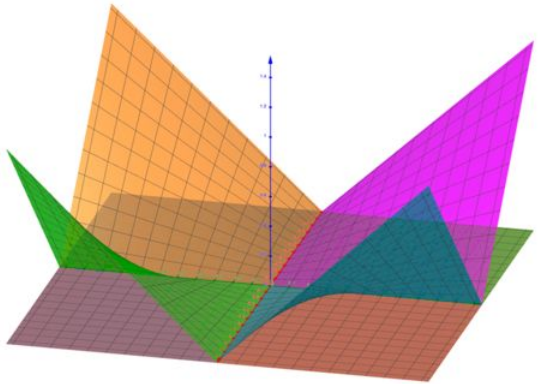


Figure 1. A visualisation of a DPFP from a 2d space (the xy-plane) to a 4d space (the four colored surfaces). Each surface is a partial function which represents one element of the 4d vector.

Implementation: One-Line Efficiency

Symmetry Enforcement

Capacity Scaling

```python
import torch
from torch import cat
from torch.nn.functional import relu as r

def dpfp(x, nu=1):
  x = cat([r(x), r(-x)], dim=-1)
  x_rolled = cat([x.roll(shifts=j, dims=-1)
            for j in range(1,nu+1)], dim=-1)
  x_repeat = cat([x] * nu, dim=-1)
  return x_repeat * x_rolled
```

Listing 1. Simple PyTorch implementation of DPFP-ν (Eq. 37).

- Complexity: Constant in space, linear in time
- Highly parallelizable
- No random sampling (unlike FAVOR+)
- Deterministic and stable

Georgia Tech

From Adding to Editing: The Delta Rule

Key idea: memory becomes editable, not just additive

Standard Rule	Delta Rule
$W^i = W^{i-1} + v^i \otimes \phi(k^i)$	$W^i = W^{i-1} + \beta^i \left(v^i - \overline{v}^i\right) \otimes \phi(k^i)$
• Adds new information • Never removes old information	• Corrects previous memory via error signal

Accumulation → Correction

Georgia Tech

Learning When to Forget: β^i

- $\beta^i = \sigma(W_\beta x^i)$
- Learned "write-strength"
- Differentiable end-to-end
- Model learns when to update vs preserve information

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

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

- $\beta \approx 1$ → overwrite memory
- $\beta \approx 0$ → keep old memory

Stability via Sum Normalization

- Standard Attention Normalization can grow unstable over long sequences
- **Sum Normalization:** Divide by the sum of components
- Ensures stable attention weights over time
- Prevents values from growing unbounded over long sequences

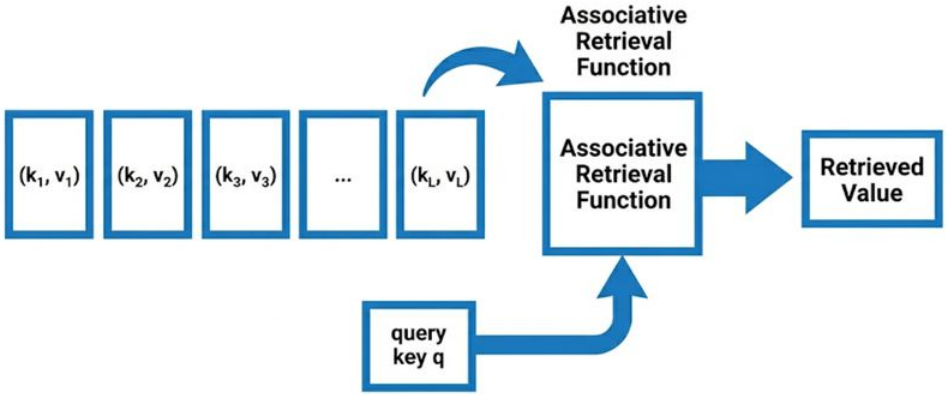
$$\phi'(\boldsymbol{q}^{(i)}) = \frac{\phi(\boldsymbol{q}^{(i)})}{\sum_{j=1}^{d_{\text{dot}}} \phi(\boldsymbol{q}^{(i)})_j} \tag{29}$$

Without this, the model becomes numerically unstable

Georgia Tech

Synthetic Retrieval: Testing the Limits

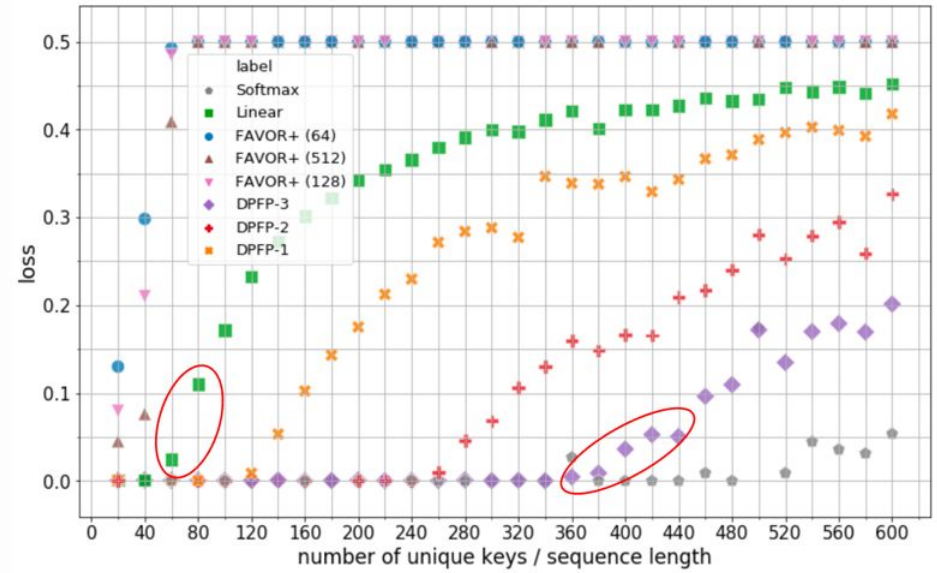
- Goal: retrieve correct value given query key
- **Setting 1** (test memory capacity)**:** Retrieve values from a sequence of length L
- **Setting 2** (test ability to update values)**:** Multiple keys re-assigned to new values; retrieve the *most recent*

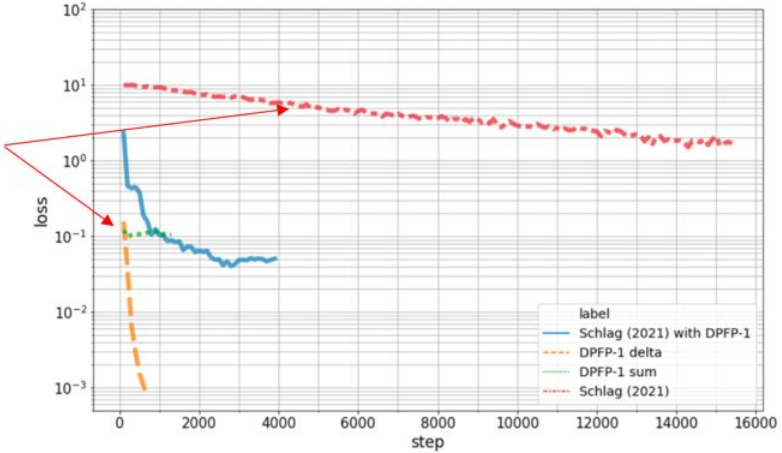


Georgia Tech

Results: Solving the Capacity Gap

- Linear Attention fails once $S > d_{key}$ (number of associations > memory dimension)
- DPFP shifts this limit → improves capacity
- Delta update rule outperforms all others on re-assignment tasks





Georgia Tech

Real-World Scaling: Machine Translation

- Task: English to German
- DPFP provides a better trade-off between simplicity and performance

Table 1. WMT14 En-De Translation BLEU scores for various Transformer models. Neither model averaging, nor model specific tuning is done. *Standard* denotes the basic Transformer.

	Valid			Test		
d_{dot}	64	256	512	64	256	512
Standard	26.6	-	-	**27.7**	-	-
Linear	25.5	-	-	26.8	-	-
Performer	24.2	24.9	**26.7**	24.4	25.3	**27.7**
DPFP (ours)	-	26.2	26.2	-	26.9	27.1

Georgia Tech

State-of-the-Art Language Modeling

- **Delta Network** = Linear Transformer + Delta Rule
- Overcapacity = sequence length > memory capacity
- Significant improvements in 'overcapacity' regimes
- Delta rule significantly improves perplexity in this regime

	Update Rule	*small*		*medium*	
		Valid	Test	Valid	Test
Transformer	-	33.0	34.1	27.9	29.6
Linear Transformer	sum	37.1	38.3	31.1	33.0
Delta Network	delta	**34.1**	**35.5**	**29.7**	**31.5**
Performer	sum	39.0	39.6	32.2	33.8
	delta	**36.1**	**37.2**	**30.0**	**31.8**

Georgia Tech

Towards Long-Context Modeling

- Delta Net is memory-efficient (State size: 0.13M)
- Maintains stability over arbitrary lengths without truncating context
- Maintains stable performance over time
- *Capacity is still bounded by feature dimension

Model	Prms. in M.	State size in M.	Perplexity Valid	Perplexity Test
Linear Transformer	89.8	0.13	>260	>260
Delta Network	89.9	0.13	27.8	29.4
Transformer-XL	90.9	0.13	65.7	65.5
		1.05	29.3	30.1
		2.10	26.4	27.4
		6.29	24.6	25.5

Georgia Tech

Shortcomings and Future Directions

- **Retrieval Precision:** Still not as 'sharp' as Softmax attention
- **The Delta-Rule Lag:** Requires re-computing weights in the backward pass for efficiency
- **Fixed Size:** Even with DPFP, there is still an upper bound on capacity

Dense Matrix

1	2	31	2	9	7	34	22	11	5
11	92	4	3	2	2	3	3	2	1
3	9	13	8	21	17	4	2	1	4
8	32	1	2	34	18	7	78	10	7
9	22	3	9	8	71	12	22	17	3
13	21	21	9	2	47	1	81	21	9
21	12	53	12	91	24	81	8	91	2
61	8	33	82	19	87	16	3	1	55
54	4	78	24	18	11	4	2	99	5
13	22	32	42	9	15	9	22	1	21

Sparse Matrix

1	.	3	.	9	.	3	.	.	.
11	.	4	.	.	.	.	.	2	1
.	.	1	.	.	.	4	.	1	.
8	.	.	.	3	1	.	.	.	.
.	.	.	9	.	.	1	.	17	.
13	21	.	9	2	47	1	81	21	9
.	.	.	.	.	.	.	.	.	.
.	.	.	.	19	8	16	.	.	55
54	4	.	.	.	11	.	.	.	.
.	.	2	.	.	.	.	22	.	21

Georgia Tech

Questions

Georgia Tech