Stick-breaking Attention

~~Stick-breaking Attention~~

^ old title…

Revised title…

Published as a conference paper at ICLR 2025

Scaling Stick-Breaking Attention: An Efficient Implementation and In-depth Study

Shawn Tan
MIT-IBM Watson AI Lab
shawntan@ibm.com

Yikang Shen
MIT-IBM Watson AI Lab
yikang.shen@ibm.com

Songlin Yang
MIT
yangsl66@mit.edu

Aaron Courville
Mila, Université de Montréal
courvila@mila.quebec

Rameswar Panda
MIT-IBM Watson AI Lab
rpanda@ibm.com

Overview

1. Prior work
2. Motivation
3. Formulation of Stick-breaking
4. Experimental results
5. Implementation details

Prior work…

THE NEURAL DATA ROUTER: ADAPTIVE CONTROL FLOW IN TRANSFORMERS IMPROVES SYSTEMATIC GENERALIZATION

Róbert Csordás[1] **Kazuki Irie**[1] **Jürgen Schmidhuber**[1,2]
[1]The Swiss AI Lab, IDSIA, University of Lugano (USI) & SUPSI, Lugano, Switzerland
[2]King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia
{robert, kazuki, juergen}@idsia.ch

2.2 GEOMETRIC ATTENTION: LEARNING TO ATTEND TO THE CLOSEST MATCH (HORIZONTAL FLOW)

We propose *geometric attention* designed to attend to the closest matching element. Like in regular self-attention, given an input sequence $[\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(N)}]$ with $\boldsymbol{x}^{(i)} \in \mathbb{R}^{d_{\text{in}}}$, each input is projected to key $\boldsymbol{k}^{(i)} \in \mathbb{R}^{d_{\text{key}}}$, value $\boldsymbol{v}^{(i)} \in \mathbb{R}^{d_{\text{value}}}$, query $\boldsymbol{q}^{(i)} \in \mathbb{R}^{d_{\text{key}}}$ vectors, and the dot product is computed for each key/query combination. In our geometric attention, the dot product is followed by a sigmoid function to obtain a score between 0 and 1:

$$\boldsymbol{P}_{i,j} = \sigma(\boldsymbol{k}^{(j)\top}\boldsymbol{q}^{(i)}) \tag{6}$$

which will be treated as a probability of the key at (source) position j matching the query at (target) position i. These probabilities are finally converted to the attention scores $\boldsymbol{A}_{i,j}$ as follows:

$$\boldsymbol{A}_{i,j} = \boldsymbol{P}_{i,j} \prod_{k \in \mathbb{S}_{i,j}} (1 - \boldsymbol{P}_{i,k}) \tag{7}$$

ModuleFormer: Modularity Emerges from Mixture-of-Experts

Yikang Shen*
MIT-IBM Watson AI Lab

Zheyu Zhang
Tsinghua University

Tianyou Cao
Tsinghua University

Shawn Tan
Mila/University of Montreal

Zhenfang Chen
MIT-IBM Watson AI Lab

Chuang Gan
MIT-IBM Watson AI Lab

3.2 Stick-breaking Self-Attention head

The stick-breaking self-attention is designed for the Transformer decoder to model the attention of each token $\mathbf{x}_t$ to previous tokens $\mathbf{x}_{<t}$. It uses the stick-breaking process view of the Dirichlet process to model the attention distribution instead of the softmax in a standard attention layer. The motivation to pay attention to the latest matching tokens. It can also be considered a simplification of the geometric attention proposed in Csordás et al. [2021].

Given an input vector sequence of t time steps $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t$, each input is projected to a sequence of key vectors $\mathbf{k}_1, \mathbf{k}_2, ..., \mathbf{k}_t$ and a sequence of value vectors $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_t$. To compute the attention of time step t, the input $\mathbf{x}_t$ is projected to a query vector $\mathbf{q}_t = \mathbf{W}_q \mathbf{x}_t$, where $\mathbf{W}_q$ is the query projection matrix. For all previous steps and the current step $i \leq t$, we compute the probability that the key at time step i matches the query at time step t:

$$\beta_{i,t} = \text{sigmoid}(\mathbf{k}_i^\top \mathbf{q}_t). \tag{3}$$

To get the attention weights of the most recent matching key, we use the stick-breaking process:

$$p_{i,t} = \beta_{i,t} \prod_{i<j\leq t} (1 - \beta_{j,t}). \tag{4}$$

THE NEURAL DATA ROUTER: ADAPTIVE CONTROL FLOW IN TRANSFORMERS IMPROVES SYSTEMATIC GENERALIZATION

Róbert Csordás[1] **Kazuki Irie**[1] **Jürgen Schmidhuber**[1,2]
[1]The Swiss AI Lab, IDSIA, University of Lugano (USI) & SUPSI, Lugano, Switzerlar
[2]King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arab
{robert, kazuki, juergen}@idsia.ch

ModuleFormer: Modularity Emerges from Mixture-of-Experts

Yikang Shen*
MIT-IBM Watson AI Lab

Zheyu Zhang
Tsinghua University

Tianyou Cao
Tsinghua University

Shawn Tan
Mila/University of Montreal

Zhenfang Chen
MIT-IBM Watson AI Lab

Chuang Gan
MIT-IBM Watson AI Lab

NEURAL LANGUAGE MODELING BY JOINTLY LEARNING SYNTAX AND LEXICON

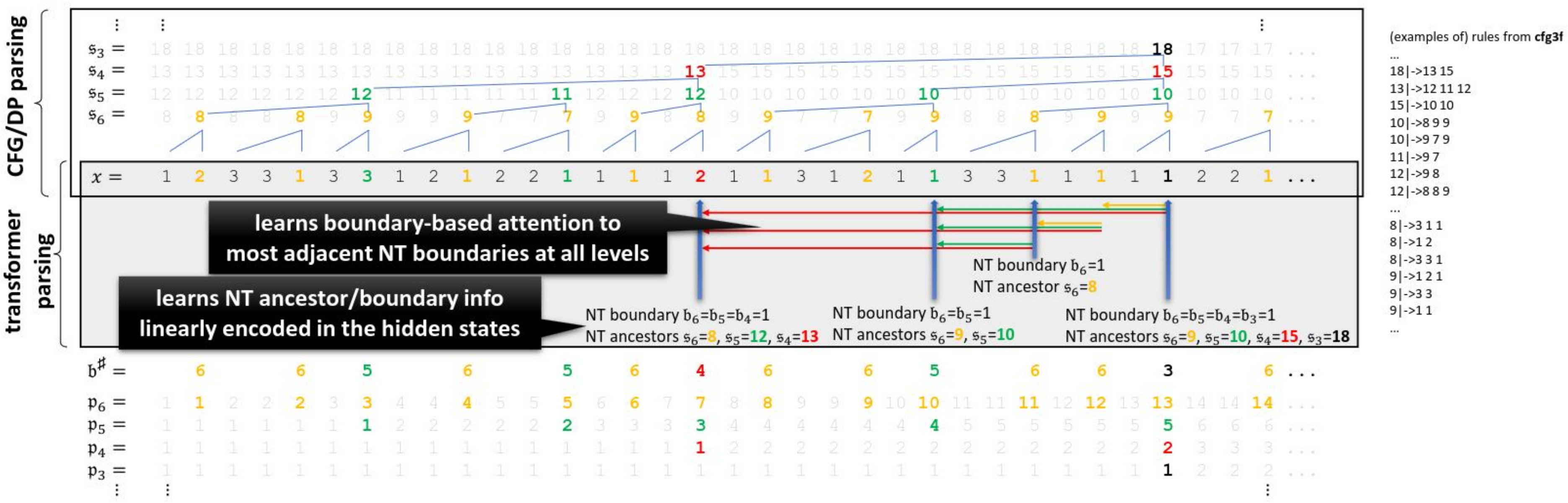
Yikang Shen, Zhouhan Lin, Chin-Wei Huang & Aaron Courville
Department of Computer Science and Operations Research
Universit de Montral
Montral, QC H3C3J7, Canada
{yi-kang.shen, zhouhan.lin, chin-wei.huang, aaron.courville}@umontreal.ca

4.1 MODELING LOCAL STRUCTURE

In this section we give a probabilistic view on how to model the local structure of language. A detailed elaboration for this section is given in Appendix B. At time step t, $p(l_t|x_0, ..., x_t)$ represents the probability of choosing one out of t possible local structures. We propose to model the distribution by the Stick-Breaking Process:

$$p(l_t = i|x_0, ..., x_t) = (1 - \alpha_i^t) \prod_{j=i+1}^{t-1} \alpha_j^t \tag{4}$$

Grammar Structures



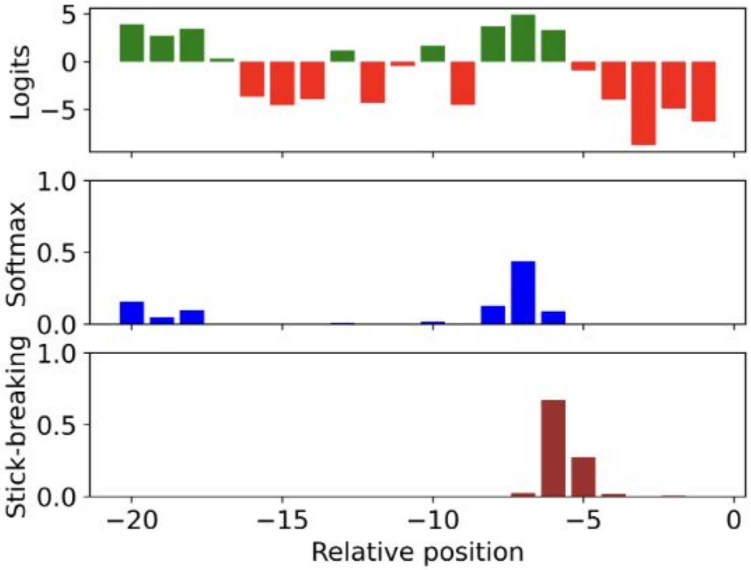
https://physics.allen-zhu.com/part-1

Stick-breaking Attention

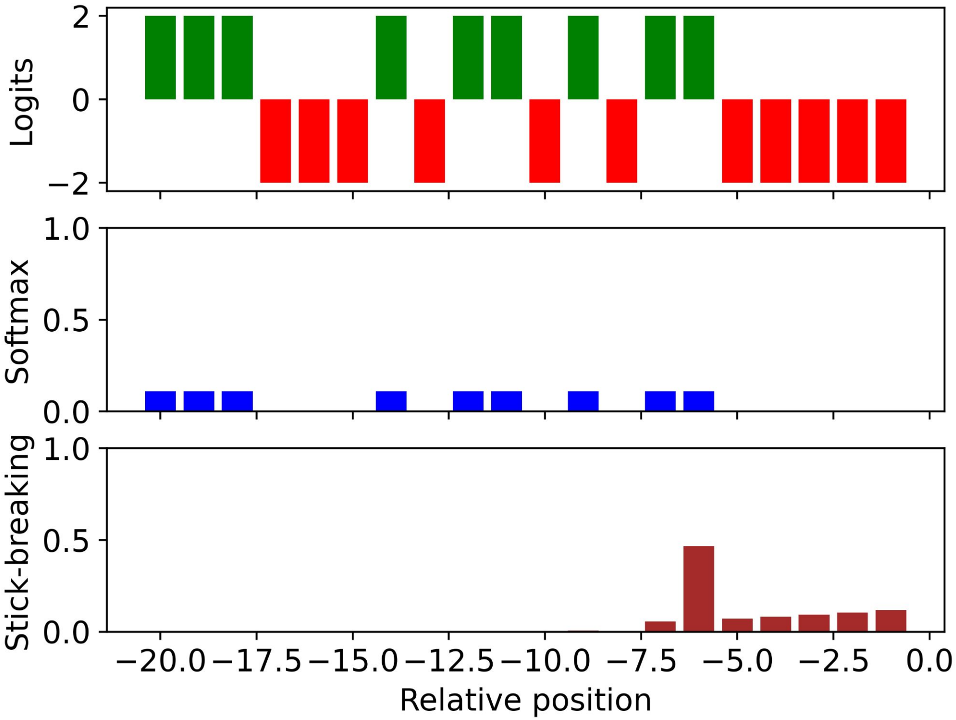
Logits $z_{i,j} = \frac{\boldsymbol{q}_i^\top \boldsymbol{k}_j}{\sqrt{d_{\text{head}}}}$

Softmax $\boldsymbol{A}_{i,j} = \frac{\exp(z_{i,j})}{\sum_{k=1}^{j} \exp(z_{k,j})}$

Stick-breaking $\boldsymbol{A}_{i,j} = \sigma(z_{i,j}) \prod_{i<k<j} (1 - \sigma(z_{k,j}))$



Stick-breaking Attention



Stick-breaking Attention

- Pros:
  - No position embeddings needed
  - Good length extrapolation behaviour
  - Possible conditional computation tricks for speedups

- Cons:
  - Computation of log-sigmoids much much slower than exponents in softmax

Experimental Results

- Small Synthetic Task (MQRAR)
- 350M models (Length Extrapolation)
- 1B & 3B models
  (general LLM evals, RULER)

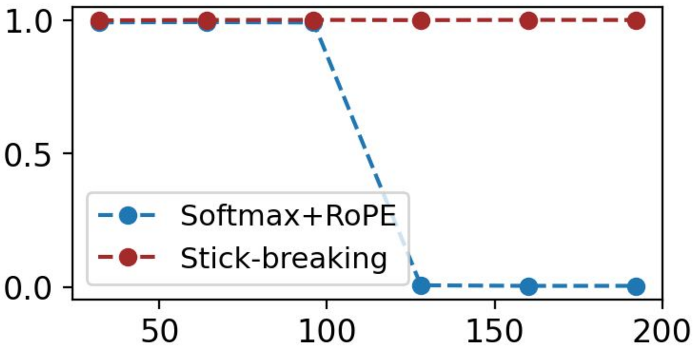
Multi-Query *Repeated* Associative Recall

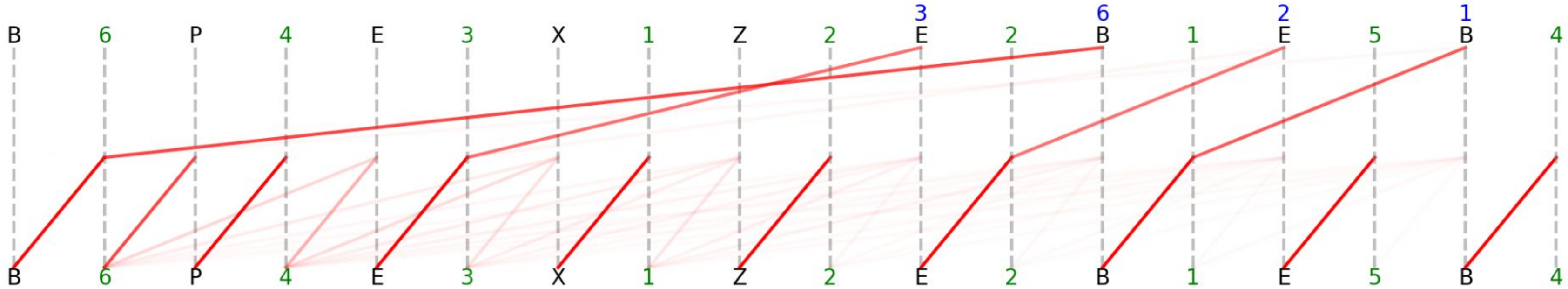
- Setting where variable is repeatedly assigned instead of assigned once
- Task is to retrieve the last assignment

Input	B	6	P	4	E	3	X	1	Z	2	E	2	B	1	E	5	B	4
Output	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	3	ϕ	6	ϕ	2	ϕ	1	ϕ

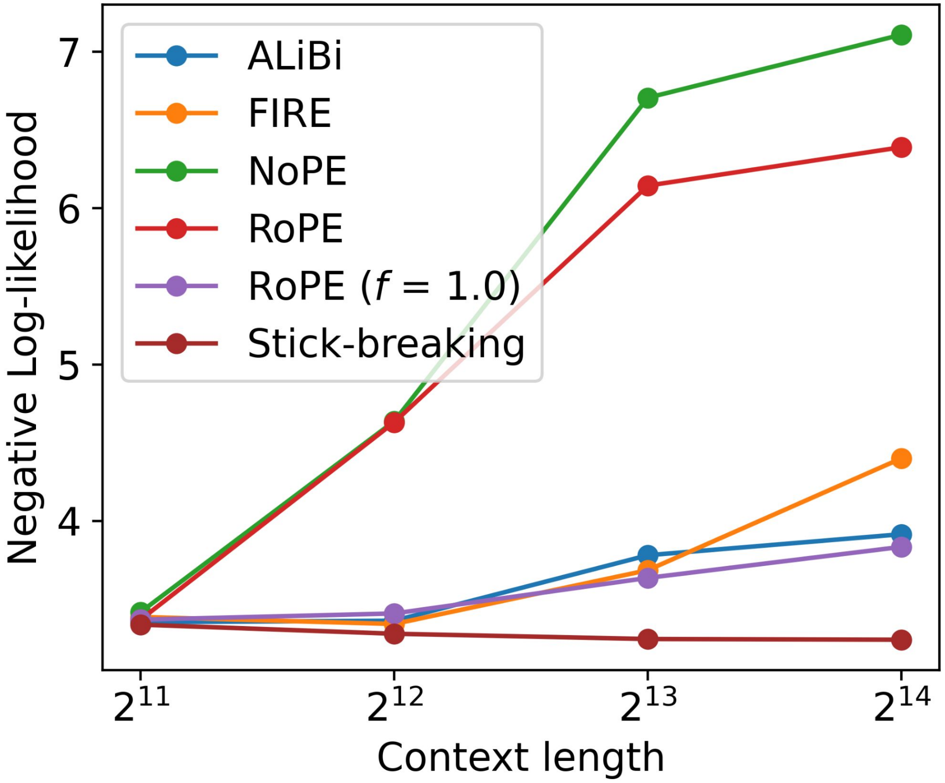
Zoology, https://arxiv.org/abs/2312.04927

Multi-Query *Repeated* Associative Recall





Length Generalisation



- 350M models
- Trained on 2048 context length
- Evaluated on longer contexts

1B and 3B model results

Task	ARC-c	ARC-e	Hella.	OBQA	PIQA	RACE	SciQ	Wino.	Avg.	Wiki.
	Accuracy (normalised)				*Accuracy*					*Ppl.*
1B Parameter Models										
Softmax	35.8	65.6	64.8	**38.8**	75.0	36.5	90.5	**63.4**	58.8	13.8
Stick-breaking	**37.7**	**67.6**	**65.4**	36.6	**76.0**	**37.4**	**91.9**	63.1	**59.5**	**13.4**
3B Parameter Models										
Softmax	42.2	73.1	73.2	**40.8**	78.8	37.4	93.5	67.6	63.3	11.3
Stick-breaking	**44.9**	**74.3**	**74.1**	40.4	**79.7**	**37.8**	**93.9**	**68.0**	**64.1**	**10.8**
Gemma2-2B	50.0	80.2	72.9	41.8	79.2	37.3	95.8	68.8	65.8	13.1
Qwen1.5-4B	39.6	61.5	71.4	40.0	77.0	38.2	90.0	68.1	60.7	12.5

1B and 3B model results

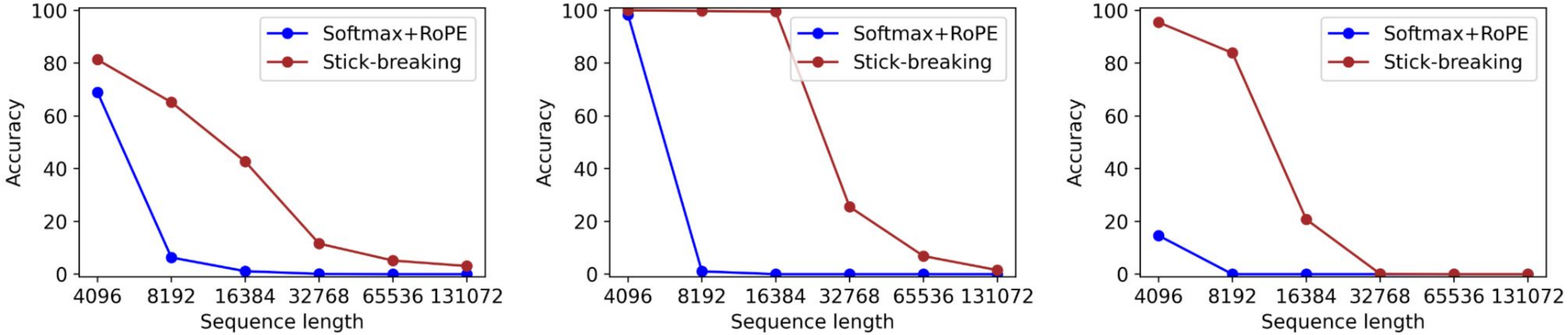
Table 4: 3B Model GSM8K Results

	GSM8K	
	5-shot	8-shot, CoT
Softmax	**44.1**	44.2
Stick-breaking	42.3	**49.7**

Table 3: MMLU few-shot results

	MMLU	
	0-shot	5-shot
1B Parameter Model		
Softmax	25.7	25.2
Stick-breaking	**28.4**	**29.3**
TinyLlama	25.3	26.0
3B Parameter Model		
Softmax	46.1	49.1
Stick-breaking	**50.8**	**52.9**
Gemma2-2B	49.3	53.1
Qwen1.5-4B	54.2	55.2

RULER benchmarks



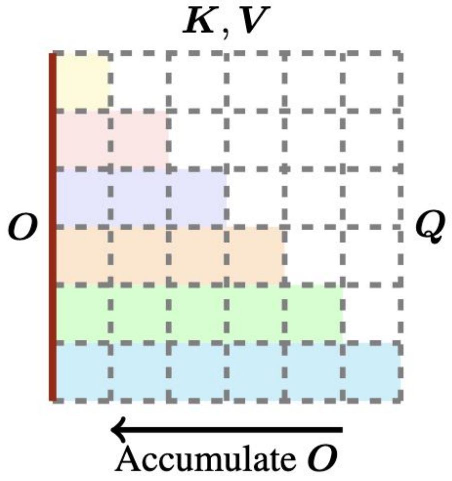
(a) Overall (b) Needle in a Haystack (NIAH) (c) Variable Tracking

Forward Pass

Forward Computing Equation 1 directly will result in underflow issues, especially with lower precision training. We perform the operations in log-space, which results in a cumulative sum instead:

$$\boldsymbol{A}_{i,j} = \exp\left(\log \beta_{i,j} + \sum_{k=i+1}^{j-1} \log\left(1 - \beta_{k,j}\right)\right) = \exp\left(z_{i,j} - \sum_{k=i}^{j-1} \log\left(1 + \exp(z_{k,j})\right)\right) \quad (4)$$

- Compute in log-space (base 2, faster)
- Skips one extra softplus computation
- Accumulates from right to left (structure of stick-breaking)
- Red border on the left denotes cumulative softplus term



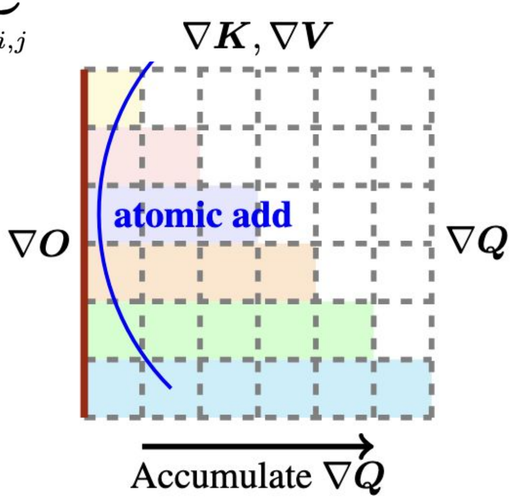
(a) Forward pass

Backward Pass

Backward Let $\tilde{\boldsymbol{A}}_{i,j} = \log \boldsymbol{A}_{i,j}$, then:

$$\frac{\partial \mathcal{L}}{\partial \tilde{\boldsymbol{A}}_{i,j}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{A}_{i,j}} \cdot \boldsymbol{A}_{i,j}, \qquad \frac{\partial \mathcal{L}}{\partial z_{i,j}} = \underbrace{\frac{\partial \mathcal{L}}{\partial \tilde{\boldsymbol{A}}_{i,j}}}_{\text{Contribution from } i,j} - \underbrace{\sigma(z_{i,j}) \sum_{i'=1}^{j-1} \frac{\partial \mathcal{L}}{\partial \tilde{\boldsymbol{A}}_{i',j}}}_{\text{Contribution from before } i,j} \tag{6}$$

- Left to right accumulation of logit gradients
- Unlike softmax, can't accumulate towards K and V
- Instead of 1 atomic add for Q gradients, it's 2 for K and V



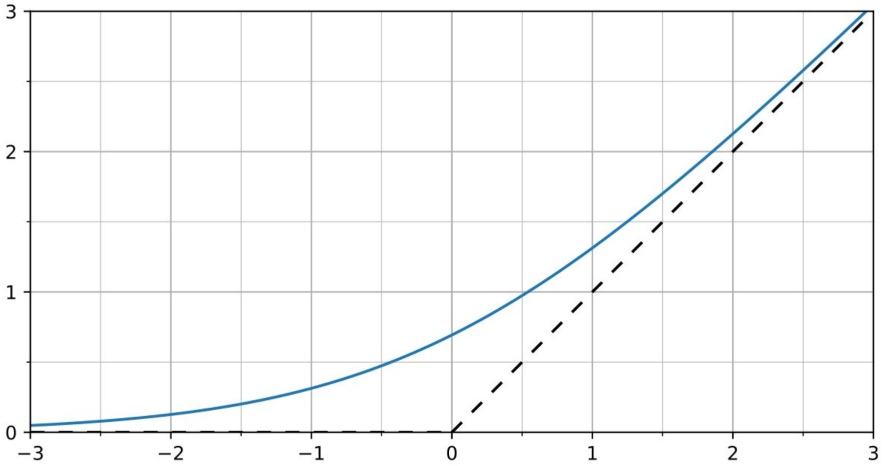
(b) Backward pass

Triton Implementation

- **Softplus acceleration**
  Naive triton implementation is too slow, used inline ASM for speedups

- **Manual `atomic_add`**
  Naive atomic add implementation slow, implemented manual while-lock

Softplus



$$\text{softplus}(x) = \begin{cases} \log\left(1 + \exp(x)\right), & \text{if } x \leq 15 \\ x & \text{otherwise} \end{cases}$$

Triton:

```
tl.where(
 x < 15.0,
 tl.math.log2(1 + tl.math.exp2(x)),
 x
)
```

Equivalent PTX:

```
.reg .pred p;
setp.gt.f32  p, ${in_reg}, 15.;
@p  mov.f32  ${out_reg}, ${in_reg};
@!p ex2.approx.ftz.f32 ${out_reg}, ${in_reg};
@!p add.f32            ${out_reg}, ${out_reg}, 1.0;
@!p lg2.approx.ftz.f32 ${out_reg}, ${out_reg};
```

While-lock for atomic add

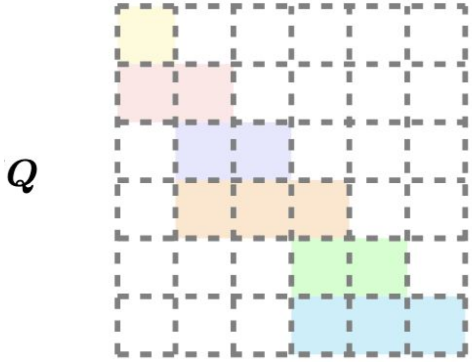
- Use HBM variable as lock
- One-lock for entire block atomic add for both k and v blocks

Source:
https://triton-lang.org/main/getting-started/tutorials/05-layer-norm.html

```
while tl.atomic_cas(Lock_ptr, 0, 1) == 1:
    pass
count = tl.load(Count_ptr)
if count == 0:
    tl.store(Count_ptr, 1)
else:
    a += tl.load(A_ptrs)
    b += tl.load(B_ptrs)
tl.store(A_ptrs, a)
tl.store(B_ptrs, b)
tl.atomic_xchg(Lock_ptr, 0)
```

Block Skipping during Decoding

- If sum of attention weights sum to 1, following time-steps can be skipped.
- Implementation caveats:
  - Block must all sum to 1 in order to do early exit
  - Have to wait for slowest head for improvements



(c) Block skipping

Additional follow-up improvements

- Works better at larger head sizes: 128 dim
- Per-head normalisation
- Remainder bias

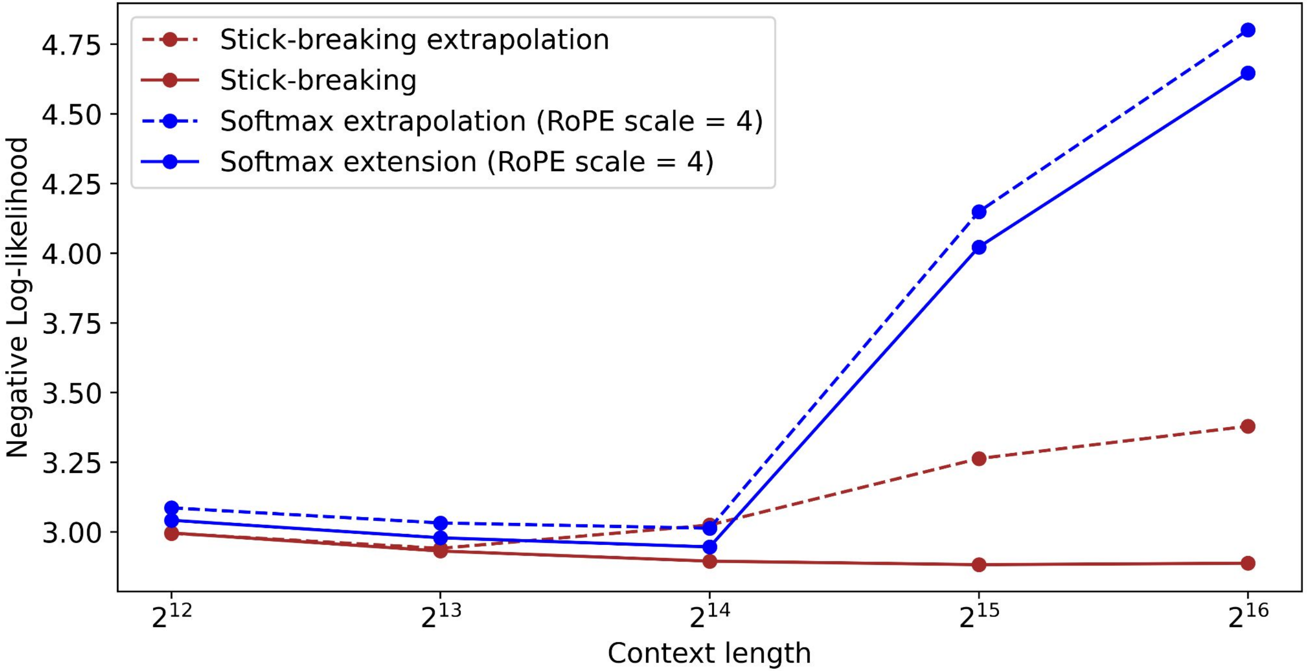
Remainder bias

- Sort of like an ‘attention sink’ – weighted with remaining mass of attention

- Maintains magnitude of output vector instead of giving almost 0 if attention weights are close to 0

$$\boldsymbol{o}_j = \sum_{i=1}^{j-1} \boldsymbol{A}_{i,j} \cdot \boldsymbol{v}_i + \left(1 - \sum_{i=1}^{j-1} \boldsymbol{A}_{i,j}\right) \cdot \boldsymbol{r}$$

Length extension



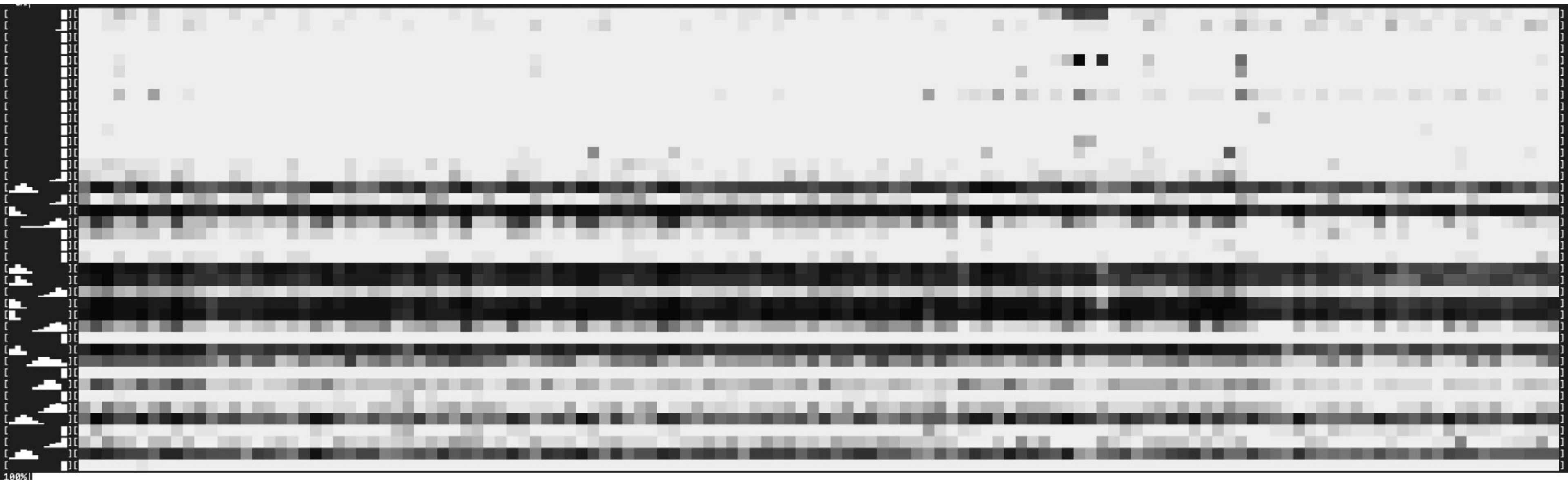
Forget Gate

- Modulate the betas in stick-breaking with a forget gate,
- Forget gate is computed only on the key-value side,
- Enables further sparsity
  -> more space in KV cache

$$\beta_{i,j} = f_i \cdot \hat{\beta_{i,j}}$$

Forget Gate

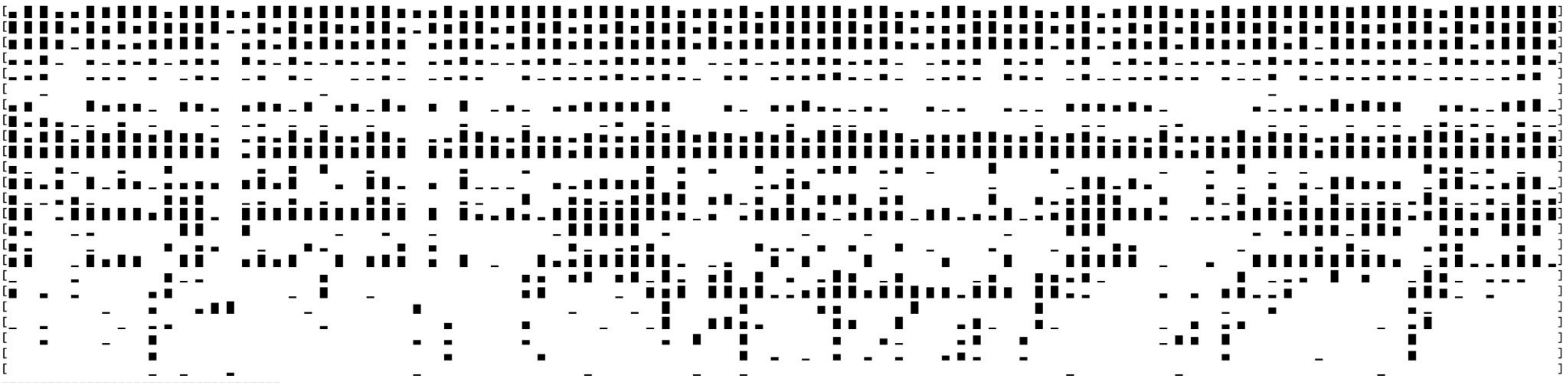
First Layer



Last Layer

Forget Gate with auxiliary loss

First Layer



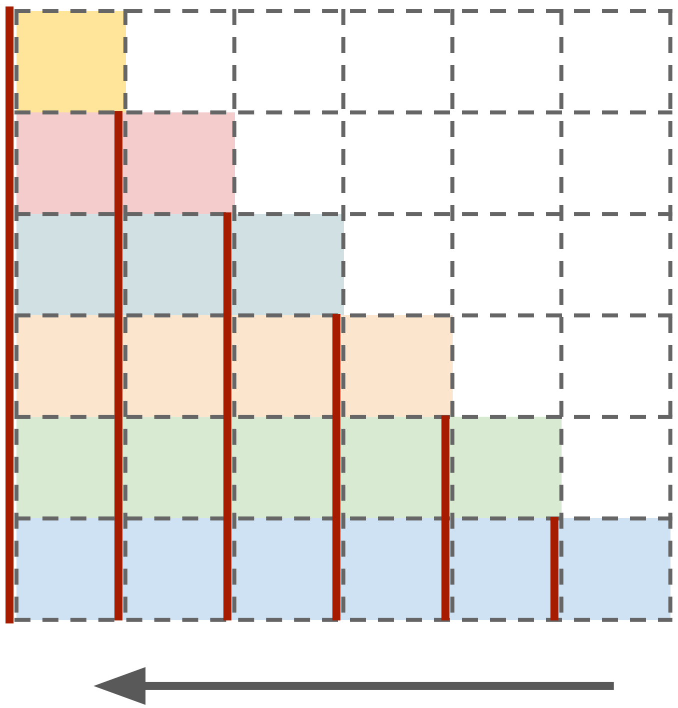
Last Layer

Recap

- **Stick-breaking attention:** swaps out softmax for stick-breaking process
- **Empirical results:** surprising length-generalisation properties, despite recency bias.
- **Triton flash-attention-like implementation:** slower, but allows scaling up

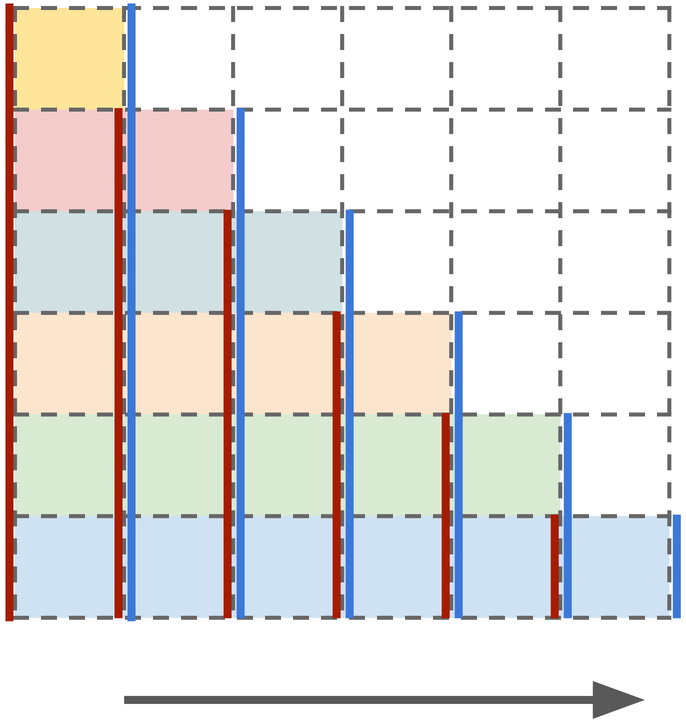
1. Forward pass

1. Each (batch, head, horizontal block) assigned to different thread.
   (Colour on right represent separate threads)
2. Calculate stickbreaking output from right to left.
3. Save cumulative log-probabilities
   (Red borders)



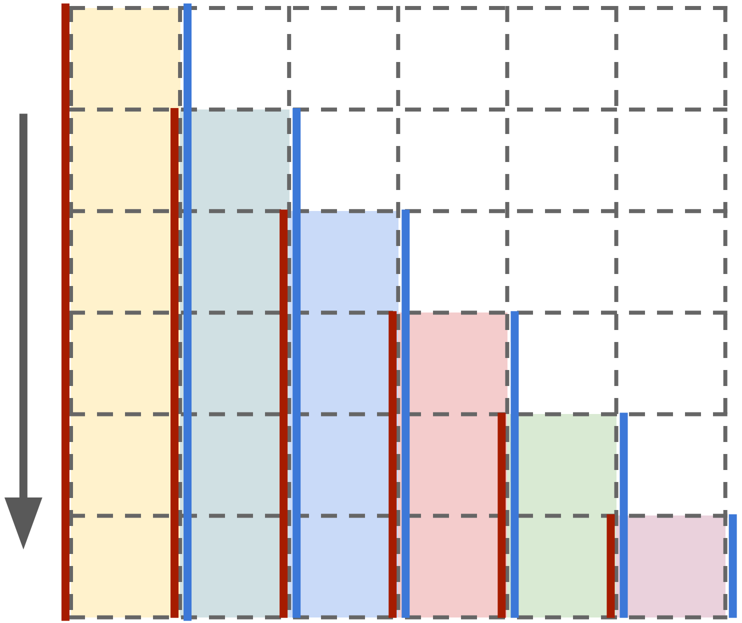
2. Backward pass (Step 1, dQ)

1. Each (batch, head, horizontal block) assigned to different thread. (Colour on right represent separate threads)
2. Calculate from left to right
3. Recompute attention probabilities using memoized values (Red borders)
4. Calculate cumulative gradients forward (blue borders)



2. Backward pass (Step 1, dV & dK)

1. Each (batch, head, vertical block) assigned to different thread. (Colour on right represent separate threads)
2. Calculate from top to bottom
3. Recompute attention probabilities using memoized values (Red borders)
4. Recompute dV, dK gradients using memoized values (Blue borders)



Sparse computation

Computation can be skipped if blocks sum to 1.