Improving Linear Attention via Softmax Mimicry

Michael Zhang[*] **Kush Bhatia, Hermann Kumbong and Christopher Ré**
Department of Computer Science
Stanford University
`{mzhang, kushb, chrismere}@cs.stanford.edu`, `kumboh@stanford.edu`

Abstract

Linear attentions are promising methods to improve Transformer efficiency. This improved efficiency is applicable to training linear Transformers from scratch, converting finetuned Transformers into linear versions that recover task-specific performance, and converting pretrained Transformers into linear versions for downstream transfer. However, linear attentions often lag behind softmax attention in performance. To address this gap, we identify two key empirical properties of softmax attention missing in linear attentions: low-entropy "spiky" weights and dot-product monotonicity. We thus introduce Hedgehog, a learnable linear attention trained to "mimic" softmax attention by minimizing cross-entropy between attention weights. Experiments show Hedgehog significantly closes the attention performance gap. Hedgehog closes 68.6% of the gap on WikiText-103 when training 125M-parameter linear Transformers from scratch, improving upon prior linear attentions by up to 6 perplexity points (PPL), and recovers >99% of GLUE points when converting finetuned BERT models, outperforming prior methods up to 8.7 points. By "linearizing" GPT-2, Hedgehog outperforms efficient Transformer alternatives, obtaining state-of-the-art 16.7 PPL on WikiText-103.

1 Introduction

Linear attentions are promising methods for improving Transformer efficiency. By replacing standard attention's softmax of query-key dot products with the dot product of alternate feature maps (Fig. 1), linear attentions reduce attention's time and space complexity from $\mathcal{O}(n^2d)$ to $\mathcal{O}(ndd')$ for sequence length n, head dimension d, and feature map dimension d' [5, 13, 17, 22, 28]. For typical Transformer settings, *e.g.,* $d = 64$ and $n = 512$ to 32K, this quadratic-to-linear scaling can result in significant speed and memory improvements (Fig. 7). Furthermore, as drop-in alternatives to softmax attention [26], linear attentions can improve Transformer efficiency not only by training new models from scratch. They can also improve efficiency of *already trained* Transformers [12, 14], swapping attentions to convert existing Transformers into linear variants. We summarize these regimes:

- **Training-from-scratch**: training new linear attention Transformers, aiming to match standard Transformers, *e.g.,* on benchmarks like Long Range Arena (LRA) [23] and WikiText-103 [15].
- **Finetuned-conversion**: swapping the attentions of (quadratic) Transformers already finetuned on individual tasks, aiming to recover original task performance with improved efficiency [12, 14].
- **Pretrained-conversion**: doing the same as finetuned-conversion but for pretrained Transformers such as large language models (LLMs), *e.g.,* to transfer to new tasks and longer contexts.

Unfortunately, existing linear attention mechanisms typically fail to match softmax attention in modeling quality. When training from scratch, linear attentions achieve 4-6 worse perplexity (ppl) than softmax attention on standard benchmarks such as WikiText-103 [10, 11, 22], the equivalent gap between 125M and 255M Transformers [6]. When converting finetuned models, linear attention models require additional standard attention modules to close the gap [12, 14].

[*]Corresponding author.

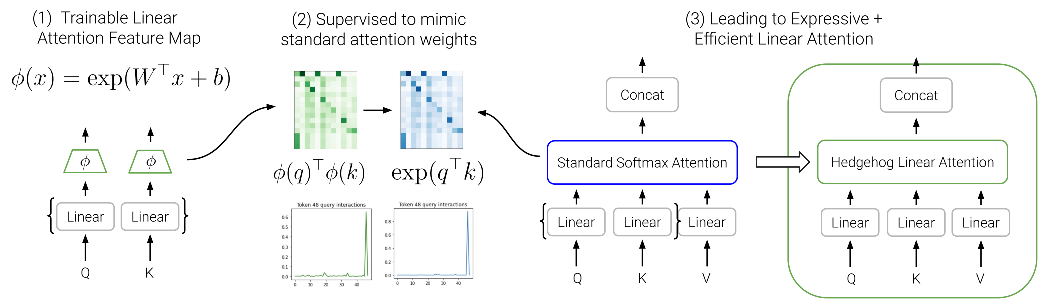
(1) Trainable Linear Attention Feature Map
$\phi(x) = \exp(W^\top x + b)$
ϕ
ϕ
Linear
Linear
Q
K
(2) Supervised to mimic standard attention weights
$\phi(q)^\top \phi(k)$
$\exp(q^\top k)$
Token 48 query interactions
Token 48 query interactions
(3) Leading to Expressive + Efficient Linear Attention
Concat
Standard Softmax Attention
Linear
Linear
Linear
Q
K
V
Concat
Hedgehog Linear Attention
Linear
Linear
Linear
Q
K
V

Figure 1: Hedgehog learns trainable feature maps so linear attention weights match those of standard attention. This results in expressive yet efficient linear attentions substitutable in various Transformers.

In this work, we aim to close the performance gap by studying prior linear attention expressivity. We identify two properties of softmax attention which prior linear attentions lack: **low-entropy "spikyness"** (Fig. 2) and **dot product monotonicity** (Fig. 3), and hypothesize these are critical for matching softmax attention performance. In App. B.1, we further show how they strongly correspond to performance in training-from-scratch (Fig. 5) and pretrained-conversion (Table 5).

We thus propose Hedgehog, an efficient-to-compute *learnable* linear attention trained to capture the spiky and monotonic softmax properties. Unlike prior works and the Taylor feature maps that propose a specific function [5, 13, 19], we learn these feature maps as single-layer MLPs specifically *trained to match* softmax attention weights. By mapping from $\mathbb{R}^d \mapsto \mathbb{R}^d$, we maintain prior linear attentions' $\mathcal{O}(nd^2)$ complexity. By training these mappings via softmax attention weights as cross-entropy soft-labels, we find Hedgehog matches softmax attention weights with much higher fidelity (Fig. 4), producing low-entropy and monotonic weights that ultimately improve linear attention quality.

In experiments, we validate that Hedgehog's improved expressivity translates to closing the softmax attention performance gap in the three regimes mentioned above:

- **Training-from-scratch**: Hedgehog matches Transformers on standard benchmarks such as Long Range Arena (LRA) [23] task, and closes the linear attention gap by 68.6% on WikiText-103.
- **Finetuned-conversion**: Hedgehog recovers >99% of original model performance on average across bidirectional encoder-only 110M parameter BERT-base models finetuned on GLUE.
- **Pretrained-conversion**: Hedgehog enables effective transfer to new tasks, outperforming modern subquadratic models by linearizing pretrained Transformers. A 125M linear GPT-2 finetuned on WikiText-103 achieves state-of-the-art 16.7 ppl for parameter-matched subquadratic models.

2 Improving linear attention via spiky and monotonic weights

2.1 Preliminaries: quadratic-to-linear complexity with linear attention

Attention setup. Let $\{\boldsymbol{q}_i\}_{i=1}^n$, $\{\boldsymbol{k}_i\}_{i=1}^n$, $\{\boldsymbol{v}_i\}_{i=1}^n$ denote the set of queries, keys, and values, with individual elements in $\mathbb{R}^d$. Let n denote sequence length and d denote head dimension. We compute attention outputs $\boldsymbol{y}_i \in \mathbb{R}^d$ by first computing similarities for each $\boldsymbol{q}_i$ and every $\boldsymbol{k}_j$ ($j \leq i$ for causal attention). Standard attention computes these similarities with the softmax over dot products [26]:

$$\boldsymbol{y}_i = \sum_{j=1}^{i} \text{sim}(\boldsymbol{q}_i, \boldsymbol{k}_j)\boldsymbol{v}_j, \quad \text{where} \quad \text{sim}(\boldsymbol{q}_i, \boldsymbol{k}_j) = \frac{\exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j / \sqrt{d})}{\sum_{m=1}^{i} \exp(\boldsymbol{q}_i^\top \boldsymbol{k}_m / \sqrt{d})} \,. \tag{1}$$

While expressive, Eq. 1 for all $\{\boldsymbol{y}_i\}_{i=1}^n$ requires $\mathcal{O}(n^2 d)$ time and memory. To more efficiently scale over long sequences, we thus want *linear attention*s that maintain standard attention's expressivity.

Linear attention and kernel feature maps. Observe that the $\exp(\cdot)$ in Eq. 1 can be viewed as a kernel function, which can be replaced in general with $\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^\top \phi(\boldsymbol{x}')$ where $\phi : \mathbb{R}^d \mapsto \mathbb{R}^{d'}$ is a feature map applied to each vector. By doing so, we can compute attention in *linear* time and space over the sequence length n [13, 25]. This is possible by rewriting Eq. 1 as:

$$\boldsymbol{y}_i = \frac{\phi(\boldsymbol{q}_i) \sum_{j=1}^{i} \left(\phi(\boldsymbol{k}_j)^\top \boldsymbol{v}_j\right)}{\phi(\boldsymbol{q}_i) \sum_{j=1}^{i} \phi(\boldsymbol{k}_j)} \,. \tag{2}$$

Prior feature maps. Numerous prior works have proposed feature maps ϕ that remain efficient while still being expressive and stable to train. These include ϕ ensuring positive attention weights, *e.g.,* via $1 + \text{ELU}$ [13] or ReLU [12], upweighting local tokens (*e.g.,* cosFormer [19]), and approximating softmax via randomized features (*e.g.,* Performer [5, 21]).

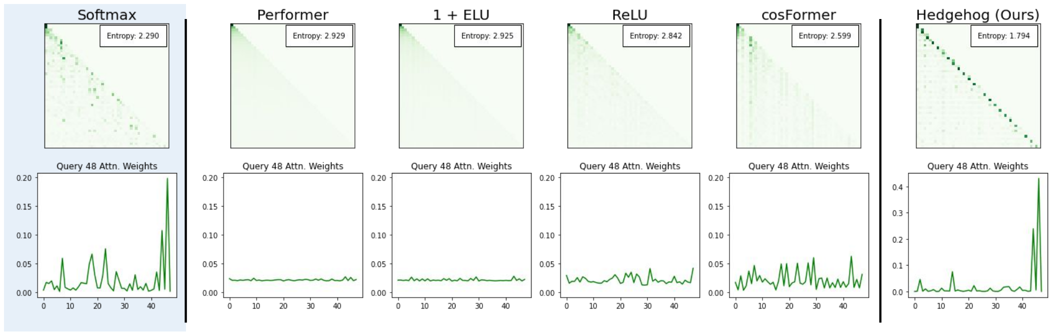

Figure 2: **Attention weight spikiness**. Softmax attention (left) display much lower entropy than prior linear attentions, resulting in "spiky" weights. Recovering this spikiness via Hedgehog (right) corresponds to improved performance (Sec. 4.1).

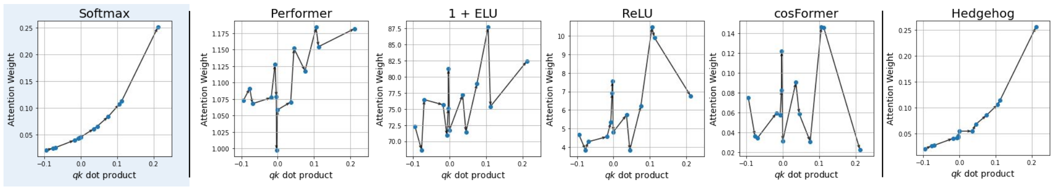

Figure 3: **Attention weight monotonicity**. Versus softmax attention (left), prior linear attentions are not monotonic over trained query-key dot products, coinciding with poor recovery during finetuned-conversion (Table 2). Hedgehog (right) can improve monotonicity and finetuned conversion (Sec. 4.2).

2.2 Explaining linear attention performance gaps via expressive attention properties

We now present our core observations, that prior linear attention feature maps fail to retain "spiky" and monotonic attentions present in softmax attention. We describe and illustrate these properties (Fig. 2, 3), and show how their absence in prior linear attentions affects attention weights in Fig. 4.

Property 1: Low-entropy spikiness. Intuitively, one source of attention's effectiveness is its selective upweighting of relevant tokens [2, 4, 9]. Mechanically, the softmax exponentiates similarities of queries and keys, quantified via low-entropy or "spiky" attention weights (Fig. 2). With prior linear attentions, the attention weights result in much higher entropy (more uniform) distributions, even for softmax approximations under mean-squared error bounds (Performer [5], Fig. 2).

Property 2: Monotonicity over query-key dot products. Beyond spikiness, we also observe that prior linear attention weights are not monotonic over trained query-key dot products (Fig. 3). We observe that while softmax attention (left subplot) has a smooth monotonic trend, none of the existing linear attentions exhibit the same monotonicity. This may cause training issues after swapping attentions due to conflicting gradients between attentions and original model parameters, *e.g.,* in Fig. 3, trying to upweight attentions by increasing product similarity can result in *lower* attention.

3 Hedgehog: Expressive linear attention via softmax mimicry

To retain these properties in an efficient linear attention, we present Hedgehog. Our key insight is that rather than use fixed functions to capture the spiky and monotonic properties, we learn linear attention feature maps as single-layer MLPs to do so.

Spiky MLP feature map. We make the feature map $\phi : \mathbb{R}^d \mapsto \mathbb{R}^{d'}$ in Eq. 2 a trainable MLP. We compute $\phi_{\text{mlp}}(\boldsymbol{q}_i)^\top \phi_{\text{mlp}}(\boldsymbol{k}_j)$ with a simple one-layer MLP as $\phi_{\text{mlp}}(\boldsymbol{x}) = \Phi(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b})$ where the matrix $\boldsymbol{W} \in \mathbb{R}^{d \times d'}$ and the bias $\boldsymbol{b} \in \mathbb{R}^{d'}$ are learned, and Φ is an activation function. For multi-head attention (and multi-layer Transformers), we use a separate MLP for each head and layer, and use the same ϕ_{mlp} for the queries and keys. To induce spikiness, we set Φ as the element-wise $\exp()$.

Attention weight distillation loss. To learn a softmax approximation, we train ϕ_{mlp} to minimize the cross-entropy loss between the computed linear attention weights and those that would have been computed via softmax attention. For query $\boldsymbol{q}_i$ and keys $\{\boldsymbol{k}_j\}_1^n$, we compute the sample losses as

$$\mathcal{L}_i = -\sum_{j=1}^{i} \frac{\exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j)}{\sum_{m=1}^{i} \exp(\boldsymbol{q}_i^\top \boldsymbol{k}_m)} \log \frac{\phi_{\text{mlp}}(\boldsymbol{q}_i)^\top \phi_{\text{mlp}}(\boldsymbol{k}_j)}{\sum_{m=1}^{i} \phi_{\text{mlp}}(\boldsymbol{q}_i)^\top \phi_{\text{mlp}}(\boldsymbol{k}_j)} \tag{3}$$

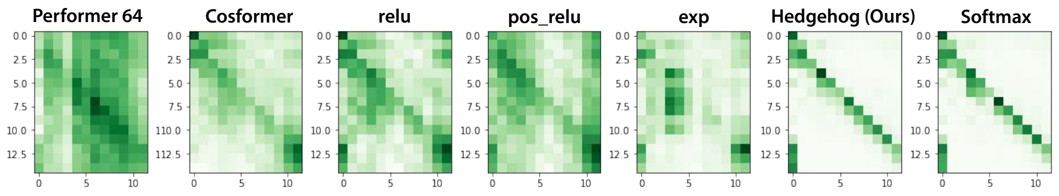

Figure 4: In contrast to prior linear attentions, trained Hedgehog attention layers (2nd right) produce attention weights that closely track softmax attention (right).

Model	Transformer	Performer	Reformer	AFT	Linear Attn. (1 + ELU)	Hedgehog
Perplexity	**18.6**	26.8	25.6	28.2	25.6	<u>20.8</u>

Table 1: Training-from-scratch on WikiText-103. Among 125M decoder-only models, Hedgehog significantly closes the gap between standard Transformers and prior linear attention maps by 68.6%.

4 Experiments

4.1 Training efficient linear Transformers from scratch

We evaluate Hedgehog Transformers trained from scratch on popular WikiText-103 language modeling [15] (also evaluating on LRA in App. C.1). Following prior work [10], we compare Hedgehog with other attentions on a 125M decoder-only Transformer, computing perplexity on 1024-token inputs. We find Hedgehog significantly closes the performance gap up to 6 points (Table 1).

4.2 Finetuned conversion of quadratic to linear transformers

We further evaluate Hedgehog for converting finetuned BERT models trained on individual GLUE tasks. We compare Hedgehog to the prior linear attention method Transformer-to-RNN (T2R) [12] in Table 2, which differs by (1) using the ReLU activation function and (2) not training to mimic softmax attention. To further test the value of Hedgehog's exponential activation function and attention distillation, we also compare against an ablation that trains the T2R feature map with our distillation loss (T2R-HH). Training to mimic softmax attentions boosts performance of T2R, suggesting that attention weight distillation may be a general step to improving linear attention. However, Hedgehog's exponential map still leads to superior performance, supporting the spiky design choice.

4.3 Pretrained conversion for subquadratic task transfer

Finally, we evaluate Hedgehog for converting pretrained Transformers into linear Transformers that can be readily finetuned on downstream tasks. We use the same WikiText-103 evaluation and 125M parameter model size as in Sec. 4.1. To evaluate Hedgehog for pretrained-conversion, we compare against T2R and other modern subquadratic sequence models such as H3 [10] and Hyena [18]. In Table 3, we find that Hedgehog outperforms T2R pretrained-conversion, H3 and Hyena.

Method	CoLA	SST2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	Avg	(%) Recover
BERT-FT	58.8	93.2	90.2	88.8	91.0	84.7	91.3	68.2	83.3	100.0
T2R	43.6	87.7	83.0	78.6	86.7	-	84.6	54.1	74.0	88.9
T2R-HH	56.9	90.9	89.1	77.7	90.0	77.4	84.5	56.3	77.9	93.5
Hedgehog	**59.2**	**92.6**	**90.1**	**87.4**	91.0	82.6	**89.6**	**69.3**	**82.7**	**99.3**

Table 2: Finetuned-conversion evaluation. Hedgehog recovers 99.3% of original finetuned BERT (BERT-FT) GLUE performance, significantly closing the gap w.r.t. prior conversion approaches.

Method	GPT-2	Hybrid H3	Hyena	T2R-GPT-2	Hedgehog-GPT-2
Perplexity	28.0	18.5	18.5	19.4	**16.7**

Table 3: Pretrained-conversion on WikiText-103. By converting GPT-2 into a linear Transformer before finetuning, Hedgehog outperforms prior competitive 125M sub-quadratic sequence models.

5 Conclusion

We study the linear attention performance gap, hypothesizing that low-entropy and monotonic attention weights are key for improving performance. We thus propose Hedgehog, a learnable linear attention to capture these properties, and show improved performance in a variety of training regimes.

References

[1] Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. *Advances in neural information processing systems*, 29, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR, 2023.

[4] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021.

[5] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[6] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1285. URL https://aclanthology.org/P19-1285.

[7] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[10] Daniel Y Fu, Tri Dao, Khaled Kamal Saab, Armin W Thomas, Atri Rudra, and Christopher Re. Hungry hungry hippos: Towards language modeling with state space models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=COZDy0WYGg.

[11] Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. Going beyond linear transformers with recurrent fast weight programmers. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=ot2ORiBqTa1.

[12] Jungo Kasai, Hao Peng, Yizhe Zhang, Dani Yogatama, Gabriel Ilharco, Nikolaos Pappas, Yi Mao, Weizhu Chen, and Noah A. Smith. Finetuning pretrained transformers into RNNs. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10630–10643, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.830. URL https://aclanthology.org/2021.emnlp-main.830.

[13] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

[14] Huanru Henry Mao. Fine-tuning pre-trained transformers into decaying fast weights. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 10236–10242, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.697. URL https://aclanthology.org/2022.emnlp-main.697.

[15] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Byj72udxe.

[16] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

[17] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A Smith, and Lingpeng Kong. Random feature attention. *arXiv preprint arXiv:2103.02143*, 2021.

[18] Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models. *arXiv preprint arXiv:2302.10866*, 2023.

[19] Zhen Qin, Weixuan Sun, Hui Deng, Dongxu Li, Yunshen Wei, Baohong Lv, Junjie Yan, Lingpeng Kong, and Yiran Zhong. cosformer: Rethinking softmax in attention. *arXiv preprint arXiv:2202.08791*, 2022.

[20] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. URL https://api.semanticscholar.org/CorpusID:160025533.

[21] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007. URL https://proceedings.neurips.cc/paper_files/paper/2007/file/013a006f03dbc5392effeb8f18fda755-Paper.pdf.

[22] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International Conference on Machine Learning*, pages 9355–9366. PMLR, 2021.

[23] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qVyeW-grC2k.

[24] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[25] Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4344–4353, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1443. URL https://aclanthology.org/D19-1443.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[27] Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019.

[28] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A nyström-based algorithm for approximating self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 14138–14148, 2021.

A Additional Hedgehog details

A.1 Implementation mechanics

We discuss additional mechanics observed to improve Hedgehog performance in practice.

Numerical stability In practice, we find that computing Φ as the softmax applied over the *MLP output dimension* also seems to work but with better stability. In this case, we expand the feature map as

$$\phi_{\text{mlp}}(\boldsymbol{x}) = \Big[\frac{\exp(\boldsymbol{w}_1^\top \boldsymbol{x})}{\sum_{i=1}^d \exp(\boldsymbol{w}_i^\top \boldsymbol{x})}, \ldots, \frac{\exp(\boldsymbol{w}_d^\top \boldsymbol{x})}{\sum_{i=1}^d \exp(\boldsymbol{w}_i^\top \boldsymbol{x})}\Big] \tag{4}$$

(also performing better than dividing each element by the max over $\{\exp(\boldsymbol{w}_i^\top \boldsymbol{x} + \boldsymbol{b})\}_{i=1}^d$)

Negation mapping. To better compute dot products as a similarity measure between queries and keys, in practice we also set Φ as a mapping from $\mathbb{R}^d \mapsto \mathbb{R}^{2d}$, *e.g.,* via

$$\phi_{\text{mlp}}(\boldsymbol{x}) = \Big[\exp(\boldsymbol{w}_1^\top \boldsymbol{x} + \boldsymbol{b}), \ldots, \exp(\boldsymbol{w}_d^\top \boldsymbol{x} + \boldsymbol{b}), \exp(-\boldsymbol{w}_1^\top \boldsymbol{x} - \boldsymbol{b}), \ldots, \exp(-\boldsymbol{w}_d^\top \boldsymbol{x} - \boldsymbol{b})\Big] \tag{5}$$

where the additional negation mapping in $\mathbb{R}^{2d}$ intuitively lets us better factor in negative dimensionalities, which prior linear attention feature maps like ReLU ignore.

A.2 Faster inference with linear attention models

Using the above equation, we can now compute the attention maps in $\mathcal{O}(ndd')$ time and space. Furthermore, while Eq. 2 is efficiently parallelizable during training (as a forward pass over n-long input sequences), [13] show that linear attention enables *constant* memory and $\mathcal{O}(n)$ decoding speed as a recurrence. If $\boldsymbol{s}_i = \sum_{j=1}^i \phi(\boldsymbol{k}_j)^\top \boldsymbol{v}_j$ and $\boldsymbol{z}_i = \sum_{j=1}^i \phi(\boldsymbol{k}_j)$, then

$$\mathbf{y}_i = \frac{\phi(\boldsymbol{q}_i)\boldsymbol{s}_i}{\phi(\boldsymbol{q}_i)\boldsymbol{z}_i}, \quad \boldsymbol{s}_i = \boldsymbol{s}_{i-1} + \phi(\boldsymbol{k}_i)\boldsymbol{v}_i^\top, \quad \boldsymbol{z}_i = \boldsymbol{z}_{i-1} + \phi(\boldsymbol{k}_i)\ . \tag{6}$$

A.3 Hedgehog feature map pseudocode

```
def hedgehog(x):
    x = linear(x) + bias
    return torch.cat([exp(x), exp(-x)])
```

B Additional motivating experiments for Section 2

B.1 Explaining linear attention performance gap

We validate the two properties introduced above by showing that (1) lacking spikiness corresponds to significantly worse performance when training from scratch, and (2) lacking spikiness and monotonicity corresponds to failing to recover performance when converting finetuned models.

Training from scratch. We compare various Transformers' abilities to solve Associative Recall (AR) [1], a next-token prediction task previously studied as a proxy for language modeling capability [16]. AR tests how well a model can recall specific content in an input sequence, structured as a list of key-value pairs which ends in a key (Table 4).

As a control for evaluating our hypothesis, we also consider a simple feature map designed to induce "spikiness" but not monotonicity: $\phi_t(x) = \exp(x \cdot t)$, which applies a temperature-t scaled exponential element-wise.

In Fig. 5, we observe a strong correspondence between low-entropy attention weights and AR task accuracy.

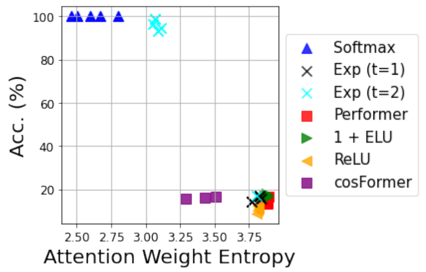

Figure 5: Associative recall performance strongly corresponds to lower attention entropy; present in softmax attention but not prior linear variants.

While softmax attention solves the AR task perfectly, prior linear attentions struggle to achieve even 20% accuracy while at the same time obtaining much larger attention map entropies. To further support our hypothesis, observe that while the exponential map ϕ_1 fails AR and produces similarly high entropy attention weights, increasing spikiness with $t = 2$ actually solves the task.

Input Sequence	Next Token	Vocab Size	Seq. Length
c $\underline{9}$ k 8 j 3 ... f 1 c	9	40	128

Table 4: Associative recall task. Example from [1].

Finetuned-conversion. We next compare how various linear attentions perform at recovering original softmax attention performance for the finetuned-conversion regime. We adopt the procedure in [12], which takes a Transformer already finetuned on a specific task, swaps the attention layers with a linear attention variant, and further finetunes the entire model on the same task.

For this setting, we evaluate with a BERT-base-uncased model [8] finetuned on the Corpus of Linguistic Acceptability (CoLA) task [27], where the goal is to classify whether a sentence is grammatically correct. We compare the performance of the original (softmax attention) BERT model[2] with the linear attention converted models. In Table 5, we find that just as no linear attention smoothly captures monotonicity over the trained model's query-key dot products, no linear attentions fully recovers the original finetuned BERT's score of 0.588. This includes the spiky ϕ_2 feature map which was sufficient in the training-from-scratch regime.

	BERT FT	1 + ELU	ReLU	Performer	cosFormer	exp(t = 1)	exp(t = 2)
MC	**58.8**	28.1	39.5	24.7	39.9	45.9	50.0

Table 5: Matthew's correlation (MC) for BERT FT on CoLA after linear attention conversion.

B.2 Architecture details

For the associative recall experiments in Sec. B.1, we use a four layer, four heads-per-layer Transformer with rotary embeddings, similar to modern model families such as Pythia [3] and LLaMA / Llama-2 [24], and keep all parts consistent except for the multi-head attention. We test various popular linear attention feature maps (c.f., Fig. 2).

B.3 Dataset details

To understand the effects of more uniform attention weightings, we evaluate with 40 possible tokens and 128 token-long-sequences, such that models must recall pairings that only occur three times on average in-context.

C Additional results and details from Section 4

C.1 Long Range Arena Results

We evaluate Hedgehog Transformers trained from scratch on the popular LRA sequence classification benchmark [23]. We initialize Transformers with Hedgehog linear attentions, and train the entire models end-to-end for sequence classification on LRA. In Table 6, we report Hedgehog results for LRA. We find Hedgehog achieves best average accuracy across tasks among linear attentions, further outperforming standard Transformers on average across the five tasks.

[2]https://huggingface.co/JeremiahZ/bert-base-uncased-cola

Table 6: Training-from-scratch on LRA. Hedgehog achieves best average performance across Transformers and subquadratic variants. * indicates method results reported from original works. All other reported from the official LRA benchmark [23]. **Best**, 2nd-best acc (%) reported.

Model	ListOps	Text	Retrieval	Image	Pathfinder	Average
Transformer	36.37	64.27	57.46	42.44	71.40	54.39
Local Att	15.82	52.98	53.39	41.46	66.63	46.06
Linear Trans.	16.13	65.90	53.09	42.34	75.30	50.55
Reformer	37.27	56.10	53.40	38.07	68.50	50.67
Sparse Trans.	17.07	63.58	59.59	44.24	71.71	51.24
Sinkhorn Trans.	33.67	61.20	53.83	41.23	67.45	51.29
Linformer	35.70	53.94	52.27	38.56	76.34	51.36
Performer	18.01	65.40	53.82	**42.77**	**77.05**	51.41
Synthesizer	36.99	61.68	54.67	41.61	69.45	52.88
Longformer	35.63	62.85	56.89	42.22	69.71	53.46
BigBird	36.05	64.02	59.29	40.83	74.87	55.01
Nyströmformer*	37.15	65.52	79.56	41.58	70.94	58.95
cosFormer*	37.90	63.41	61.36	43.17	70.33	55.23
Skyformer*	**39.25**	64.70	82.06	40.77	70.73	59.50
Hedgehog	37.15	64.60	**82.24**	40.15	74.16	**59.66**

C.2 Hyperparameters

On LRA, for fair comparison we implement Hedgehog in the existing PyTorch implementation provided by [28], using the same configs in the original repository [23]. For WikiText-103, we train 125M parameter decoder-only Transformer models with learning rate 6e-4, weight decay 0.01, and AdamW optimizer.

For the finetuned conversion setting, we first do Hedgehog attention by training attention layers for up to five epochs with learning rate 1e-2, weight decay 0, AdamW optimizer, and early stopping with respect to validation loss. We then train BERT models with learning rate 1e-5, weight decay 0, and cosine scheduler.

For the pretrained conversion setup, we start with pretrained GPT-2 [20]. We use the same distillation scheme for training Hedgehog attentions. We train Hedgehog attentions by computing queries, keys, and attention weights from forward-passes with the non-finetuned GPT-2 model on WikiText-103 samples, before swapping the attentions and finetuning the converted Hedgehog-GPT-2 model with the standard next-token prediction task. We then train all models with learning rate 6e-4, weight decay 0.01, and AdamW optimizer.

D Additional results

D.1 Hedgehog scaling

In Fig. 7 we show that Hedgehog achieves real-world linear scaling with respect to sequence length in compute and memory. We benchmark inference speed (measured in wall-clock time) and memory usage. Due to its linear scaling, we find Hedgehog achieves 4x faster inference with similar memory to FlashAttention [7] at sequence lengths 32K tokens long. For comparison, we also benchmark a 2nd-order Taylor series feature map approximation to the exponential function. While this feature map is also able to recover the low-entropy and monotonic attention weight properties that we discussed for expressive linear attentions (Fig. 6), we show that it introduces practical inefficiencies. While technically linear in sequence length, the Taylor feature maps are inefficient to compute as linear attention. They take $\mathcal{O}(nd^{p+1})$ time and space (where p is polynomial degree), resulting in non-competitive scaling compared to standard attention in wall-clock time and memory (Fig. 7).

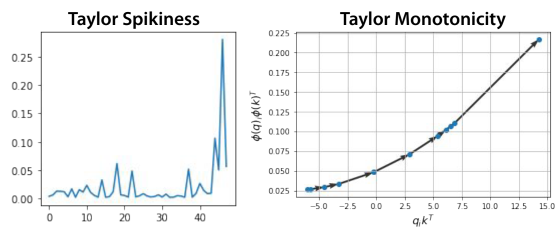

Figure 6: Taylor approximation recovers spikiness and monotonicity

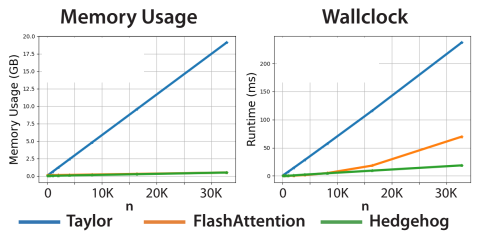

Figure 7: Hedgehog scaling in memory usage (**left**) and runtime (**right**)

D.2 Finetuned conversion attention weights

To visualize Hedgehog learned attentions throughout a model, we plot the attention weights of trained Hedgehog attentions in comparison to standard softmax attention and other linear attentions for each head and layer 0. We visualize attention weights for BERT models finetuned on the CoLA dataset, plotting the same validation set sample. Hedgehog attentions consistently best match softmax attentions qualitatively. For each layer, head, and method, we also report the average entropy over the attention weights, noting that Hedgehog attentions consistently match softmax attention's low entropy.

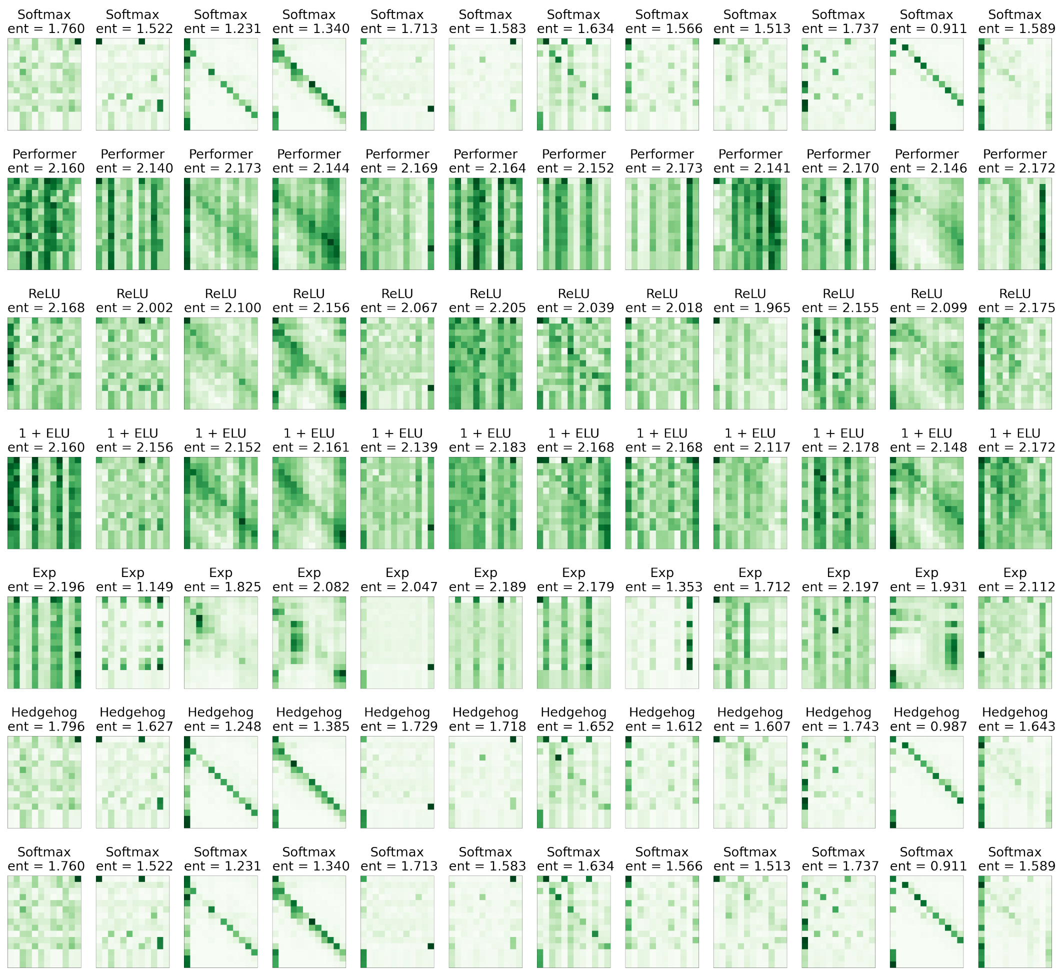

Figure 8: BERT finetuned on CoLA attention weights for layer 0.