The Role of Outgoing Connection Heterogeneity in Feedforward Layers of Large Language Models

Felix Stahlberg and Shankar Kumar
Google Research
{fstahlberg,shankarkumar}@google.com

Abstract

We report on investigations into the characteristics of outgoing connections in feedforward layers of large language models. Our findings show that inner neurons with diverse outgoing connection strengths are more critical to model performance than those with uniform connections. We propose a new fine-tuning loss that takes advantage of this observation by decreasing the outgoing connection entropy in feedforward layers. Using this loss yields gains over standard fine-tuning across two different model families (PaLM-2 and Gemma-2) for downstream tasks in math, coding, and language understanding. To further elucidate the role of outgoing connection heterogeneity, we develop a data-free structured pruning method, which uses entropy to identify and remove neurons. This method is considerably more effective than removing neurons either randomly or based on their magnitude.

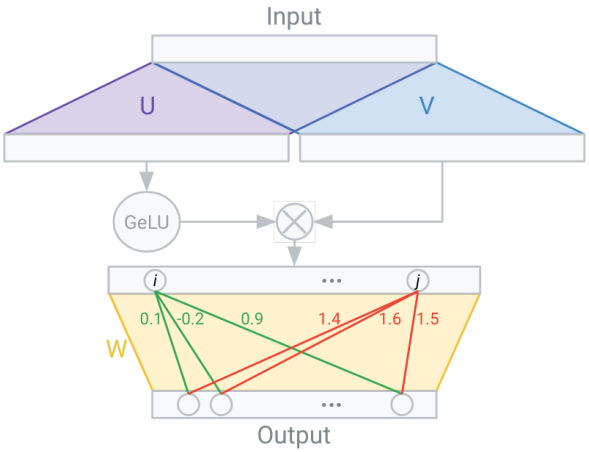

Figure 1: A GeGLU (Shazeer, 2020) feedforward layer in a Transformer-based (Vaswani et al., 2017) LLM like Gemma (Gemma et al., 2024) with incoming weight matrices U and V and outgoing weight matrix W. The neuron i with non-uniform connections is more functional than neuron j with mostly uniform connections.

1 Introduction

Neuroscience has long recognized the importance of neuronal diversity for the brain's computational power. However, this principle has largely been overlooked in the development of artificial neural networks (Fan et al., 2025). Our research demonstrates the benefits of increased heterogeneity within the outgoing connections of feedforward layers in large language models (LLMs), mirroring the functional advantages observed in biological systems. We propose a new loss called Non-Uniform Connectivity Loss (NUCL) for fine-tuning LLMs that biases the outgoing connections of inner neurons in feedforward layers towards a non-uniform, i.e. low-entropy distribution. The intuition behind NUCL is illustrated in Fig. 1. The neuron j (in red) activates all output neurons equally, and thus is unlikely to be a sharp representation of functionality, particularly as the layer is usually followed by layer normalization (Ba et al., 2016). NUCL explicitly penalizes uniform row vectors in the outgoing weight matrix W, aiming to boost the usefulness of each inner neuron. We show that PaLM-2 and Gemma-2 models in various sizes improve performance on math, coding, and language understanding tasks when fine-tuned with a loss function that combines NUCL with standard negative log-likelihood. NUCL is highly practical due to its easy implementation, requiring only a few lines of code.

To better understand the importance of connection heterogeneity in pre-trained models, we develop a data-free neuron removal procedure based on the weight matrix Ws' row vector entropies. When 25% of the neurons are removed from the Gemma feedforward layers, our entropy-based method shows the least performance degradation compared to the no-pruning baseline, outperforming random and magnitude based pruning. Surprisingly, magnitude-based pruning performs worse than random pruning.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 22476–22484
November 4-9, 2025

Our fine-tuning experiments and our pruning experiments reveal a crucial insight into how LLMs function: the performance of a model is fundamentally linked to the neuronal diversity in its feedforward layers.

2 The non-uniform connectivity loss

LLMs are commonly trained to predict the next token in a sequence. The standard loss function for a training example $\mathbf{x} = \langle x_1, \dots x_l \rangle$ of length l is the (text-based) negative log-likelihood:

$$\mathcal{L}_{\text{NLL}}(\mathbf{x}, \Theta) = -\sum_{k=1}^{l} \log P_\Theta(x_k | x_1, \dots, x_{k-1}) \tag{1}$$

where $P_\Theta(\cdot)$ is the conditional token-level probability distribution given by the LLM parameterized with Θ. The model parameters Θ contain a set $\mathcal{W}_\Theta$ of feedforward outgoing weight matrices, one for each layer in the model (cf. Fig. 1). These matrices have m (inner dimensionality) rows and n (model dimensionality) columns (i.e. $\mathcal{W}_\Theta \subset \mathbb{R}^{m \times n}$), where $m \gg n$ in Gemma models (cf. Appendix A). For a weight matrix $W \in \mathcal{W}_\Theta$ and a row $i \in [1, m]$ we define the *outgoing weight distribution* $\mathbf{r}^{W,i} \in \mathbb{R}^n$ as the normalized absolute values in the i-th row of W:

$$\mathbf{r}^{W,i} = (r_1^{W,i}, \dots, r_n^{W,i}) \text{ with } \forall j \in [1, n] : r_j^{W,i} = \frac{|W_{ij}|}{\sum_{j'=1}^{n} |W_{ij'}|}. \tag{2}$$

We propose the Non-Uniform Connectivity Loss (NUCL) that biases the $\mathbf{r}^{W,i}$ vectors towards a non-uniform distribution. Ideally, we would directly use the entropy of the outgoing weight distributions, i.e. – dropping the W, i-superscripts for clarity:

$$\mathcal{L}_{\text{NUCL-ent}}(\mathbf{r}) = -\sum_{j=1}^{n} r_j \log r_j. \tag{3}$$

In practice, however, we resort to surrogate functions due to numerical instabilities when using $\mathcal{L}_{\text{NUCL-ent}}(\cdot)$ for LLM fine-tuning. We experiment with two loss variants. The first variant (*gini*), inspired by decision tree learning, is based on the gini impurity (Breiman et al., 1984) in $\mathbf{r}$:

$$\mathcal{L}_{\text{NUCL-gini}}(\mathbf{r}) = 1 - \sum_{j=1}^{n} r_j^2. \tag{4}$$

Our second variant (*mstd*) directly optimizes the variance by minimizing the negative standard deviation of $\mathbf{r}$:

$$\mathcal{L}_{\text{NUCL-mstd}}(\mathbf{r}) = -\sqrt{\frac{\sum_{j=1}^{n} \left(r_j - \frac{1}{n}\right)^2}{n}}. \tag{5}$$

Finally, we combine NUCL linearly with the usual negative log-likelihood using a tunable scaling hyper-parameter α:

$$\mathcal{L}(\mathbf{x}, \Theta) = \mathcal{L}_{\text{NLL}}(\mathbf{x}, \Theta) + \alpha \sum_{W \in \mathcal{W}_\Theta} \sum_{i=1}^{m} \mathcal{L}_{\text{NUCL}}(\mathbf{r}^{W,i}). \tag{6}$$

A JAX implementation of NUCL is provided in Appendix D.

3 Experimental setup

We use publicly available pre-trained LLMs and evaluate on publicly available benchmarks. We explore two LLM model families – PaLM 2 (Anil et al., 2023) and Gemma 2 (Gemma et al., 2024) – each in three different sizes. For PaLM 2 we use the three sizes available via the Google Cloud API: **Gecko**, **Otter**, and **Bison**. Gemma 2 is available in **2B**, **9B**, and **27B** parameter variants (cf. Appendix A). Training details are summarized in Appendix C. We use three training datasets:

GSM8K (Cobbe et al., 2021) is a small dataset of high-quality grade school math word problems often used to assess LLMs.

SuperGLUE (Wang et al., 2019) is a benchmark consisting of eight different language understanding tasks. Training set sizes range from 250 to 101K examples which we mix proportionally, following the `super_glue_v102_proportional`[1] recipe in T5 (Raffel et al., 2020). Like prior work we report average scores across all tasks, averaging scores of tasks with multiple scores first.

MBPP (Austin et al., 2021) contains **m**ostly **b**asic **P**ython **p**roblems solvable by entry-level programmers. We report 3-shot success rates. We augmented the MBPP training partition by proprietary coding data.

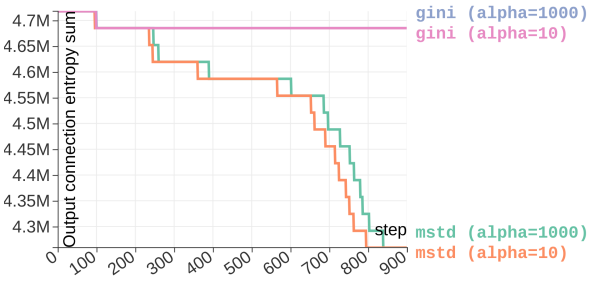

Figure 2: Output connection entropy summed over inner neurons in all feedforward layers (Gemma-9B).

Figure 3: Text-based NLL loss curve when training in combination with NUCL (Gemma-9B).

4 Results

4.1 NUCL

We begin by evaluating the performance of our *mstd* and *gini* variants. The training curves in Fig. 2 and Fig. 3 demonstrate that the training process is stable across a wide range of α-values (10 to 1000), with only minor variations in convergence for *mstd*. Notably, Fig. 2 reveals that while both variants reduce output connection entropy, *mstd* achieves significantly greater reduction. Fig. 3 indicates that *mstd* also results in better Text NLL loss. The superiority of *mstd* is further substantiated by Table 1, which shows the relative improvements over standard fine-tuning on downstream tasks.[2] While both *mstd* and *gini* show significant positive gains, *mstd* offers a greater relative improvement, achieving 3.45% on downstream tasks, compared to *gini*'s 3.07%. We will therefore use *mstd* for the rest of this paper.

Table 2 expands our evaluation to three Gemma model sizes on MBPP and GSM8K. With the exception of the 9B model on MBPP, NUCL consistently delivers relative gains ranging from 1.56% to 3.78% compared to standard fine-tuning. What makes these gains particularly noteworthy is that NUCL is very simple to implement, requiring minimal code changes.

With NUCL's effectiveness established on outgoing feedforward weight matrices, we investigated its applicability to other Transformer components. Specifically, we tested NUCL on both incoming feedforward weights (U and V in Fig. 1) and attention layer outgoing weights. However, as shown in Table 3, the most significant performance gains are achieved when NUCL is applied to outgoing feedforward weights.

Task	Model	Relative gain (in %) gini	mstd
GSM8K (accuracy)	Gecko	10.61	11.45
	Otter	2.50	2.64
	Bison	0.80	1.34
SuperGLUE (avg. score)	Gecko	1.61	1.57
	Otter	0.26	0.23
	Bison	2.64	3.47
Average		**3.07**	**3.45**

Table 1: Relative improvements over standard fine-tuning for the PaLM-2 model family.

Model size	Relative gain (in %) MBPP	GSM8K
2B	3.27	2.71
9B	-1.43	3.51
27B	1.56	3.78

Table 2: Relative improvements over standard fine-tuning for the Gemma-2 model family.

4.2 Entropy-based data-free neuron removal

Having demonstrated the performance benefits of increasing outgoing weight heterogeneity during fine-tuning, we now investigate its role in unmodified pre-trained models. To that end, we use neural network pruning as our testbed. The pruning literature is extensive (Blalock et al., 2020), with many of the most successful techniques em-

Weight matrices	Relative gain (in %)
FFN outgoing	2.71
FFN incoming (linear)	0.34
FFN incoming (gated)	0.67
Attention outgoing	-0.18

Table 3: Relative improvements over standard fine-tuning for different Transformer components (GSM8K, Gemma-2B).

[1] https://github.com/google-research/text-to-text-transfer-transformer/blob/main/t5/data/mixtures.py

[2] Relative gains over standard fine-tuning (i.e. $\alpha = 0$) are reported to facilitate a more consistent analysis across different model sizes. For absolute evaluation numbers, please refer to Appendix B. The results for the Gecko model on GSM8K may be affected by the low absolute accuracy this small baseline model achieves on this task, making the relative improvement scores more volatile.

Pruning	Method	MMLU 5-shot	HellaSwag 10-shot	ARC-e 0-shot	PIQA 0-shot	SIQA 0-shot	TriviaQA 5-shot	NQ 5-shot	Average
0%	Pre-trained baseline	52.02	73.86	80.72	78.45	51.64	60.24	17.24	**59.17**
25%	Srinivas and Babu (2015)	22.95	26.29	26.64	49.40	39.10	0.00	0.00	**23.48**
	Stahlberg and Byrne (2017)	22.95	26.72	26.05	47.66	38.23	0.00	0.01	**23.09**
	Random	26.86	52.88	62.75	70.24	47.80	8.41	2.71	**38.81**
	Magnitude-based	22.95	26.71	26.30	49.89	37.97	0.00	0.00	**23.40**
	Entropy-based	36.33	59.63	69.95	72.74	47.70	23.51	4.87	**44.96**

Table 4: Data-free removal of 25% of the neurons in the feedforward layers of Gemma-2B.

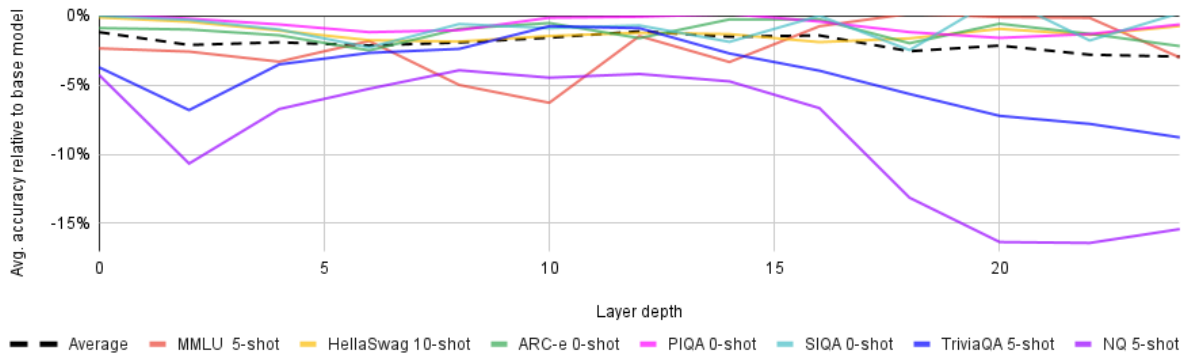

Figure 4: Downstream performance relative to the unpruned baseline (first row in Table 4) when applying entropy-based shrinking to isolated layers. Each data point represents removing 25% of feedforward neurons in two adjacent layers at a certain depth.

ploying unstructured pruning, which can result in sparse networks, and/or relying on training data for improved performance. To isolate the impact of outgoing connection heterogeneity in pre-trained models, we restrict our analysis to data-free (i.e. without any training data) structured pruning (He and Xiao, 2024) of entire neurons (as opposed to weights). An intuitive magnitude-based[3] criterion for removing neurons is to discard those with outgoing weights close to zero as they contribute relatively little to the overall layer output:[4]

$$\min_{i\in[1,m]} \sum_{j=1}^{n} W_{ij}^2. \tag{7}$$

Prior work extended this intuition with a neuron similarity criterion (Srinivas and Babu, 2015) and a compensation mechanism for the neuron removal (Stahlberg and Byrne, 2017) and showed that it is effective for convolutional and recurrent networks.

In this work, we propose to remove neurons with high outgoing connection entropies:

$$\max_{i\in[1,m]} -\sum_{j=1}^{n} r_j^{W,i} \log(r_j^{W,i}). \tag{8}$$

[3]Not to be confused with magnitude-based *weight* pruning (Han et al., 2015) that removes individual weights.

[4]We reuse the notations from Sec. 2.

We removed 25% of the inner neurons in Gemma 2B feedforward layers (16% of the full model size) by repeatedly applying the removal criterion, and then compared the pruned models on the zero- and few-shot metrics used in the Gemma-2 technical report (Gemma et al., 2024).

Surprisingly, Table 4 reveals that magnitude-based pruning and its extensions from Srinivas and Babu (2015) and Stahlberg and Byrne (2017) perform significantly worse than random selection. This suggests that the intuition behind magnitude-based pruning fails to apply to gated feedforward layers in Transformer-based LLMs. The complete performance breakdown observed on tasks like TriviaQA and NQ indicates the removal of critical neurons by these procedures.

Our entropy-based criterion – though still worse than the unpruned baseline – retains the highest level of performance among the pruned models. This underscores the crucial contribution of neurons with high outgoing weight heterogeneity to model accuracy, even in models not trained with NUCL. It also confirms prior work that found a high degree of redundancy in feedforward layers (Pires et al., 2023).

A practical question is which layers are most sensitive to pruning. Fig. 4 shows that pruning isolated layer pairs in the layer stack close to the

Layers	Average accuracy
Bottom half of the layers	51.39%
Top half of the layers	52.70%
Even layers (after global attention)	52.46%
Odd layers (after local attention)	51.84%

Table 5: Shrinking only half of the layers with entropy-based neuron removal. We report average accuracy across the tasks in Table 4.

input or close to the output has a more negative effect on certain tasks, such as TriviaQA and NQ. However, this pattern is not consistent across all tasks. Table 5 shows the average accuracies when pruning only half of the feedforward layers. Pruning the top half, bottom half, or alternating layers[5] all yield similar accuracies. This suggests that the pruning sensitivity is not tied to these simple structural groupings.

5 Related work

Our investigation extends and complements prior research on understanding the inner workings of Transformers (Räuker et al., 2023; Ferrando et al., 2024). Notably, studies have proposed that their feedforward layers act as key-value memories (Sukhbaatar et al., 2019; Geva et al., 2021). Building on this, researchers have identified neurons with distinct functional specializations within these layers, including those dedicated to factual knowledge (Dai et al., 2022), tasks (Song et al., 2024), relations (Liu et al., 2025), and language in multilingual LLMs (Kojima et al., 2024). These studies reinforce our conclusions, which indicate the benefits of diversity within feedforward layers.

It may be tempting to associate NUCL with sparsity, and draw parallels to sparsity inducing losses like L2 (Zhao et al., 2009) or sparsely connected neural networks (Han et al., 2015). However, we wish to emphasize that NUCL focuses on connection variance, not sparsity. NUCL aims to improve the functionality of individual neurons, rather than serving as a form of regularization.

6 Conclusion

Our research establishes the crucial role of heterogeneity in the feedforward outgoing weights of PaLM-2 and Gemma models. We demonstrated performance improvements in fine-tuning by promoting non-uniform outgoing weights using NUCL, our novel loss function. For existing pre-trained models, we showed that neurons with heterogeneous outgoing weights are crucial for zero- and few-shot performance; preserving them during data-free pruning minimizes performance degradation.

Limitations

We acknowledge that restricting NUCL to the fine-tuning phase is a limitation. Exploring its impact during pre-training, albeit computationally demanding, could reveal NUCL's full potential to achieve even greater reductions in outgoing weight entropy.

A further limitation is NUCL's focus on the heterogeneity of a *single* neuron's outgoing weights. To align even more with the principle of neuronal differentiation in neuroscience, which emphasizes diversity *between* neurons, a promising future direction is to expand this work to incorporate inter-neuron diversity.

Our network pruning experiments focused solely on data-free removal of entire neurons within feedforward layers, limiting our analysis of weight heterogeneity to pre-trained models. For optimal real-world performance, a light fine-tuning step after pruning would be crucial, as supported by prior research.

The definition of the outgoing weight distribution relies on the magnitudes of the weights, which intentionally disregards sign information. This design choice was made on the premise that signs can be easily inverted in subsequent layers. Future work could explore other ways to measure the heterogeneity of the outgoing weight vectors.

Working with LLMs means our research inherently carries the same ethical and societal risks as these models. This includes concerns like bias amplification and the potential for misuse, such as creating deepfakes or spreading misinformation. Our backbone models, like Gemma-2, are built to mitigate some of these risks.[6]

References

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, and 1 others. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

[5] Pruning alternating layers is motivated by the Gemma 2 architecture, which alternates between local and global attention layers.

[6] https://ai.google.dev/responsible/docs

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. 2020. What is the state of neural network pruning? In *Proceedings of Machine Learning and Systems*, volume 2, pages 129–146.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. JAX: composable transformations of Python+NumPy programs.

Leo Breiman, Jerome Friedman, Charles J Stone, and RA Olshen. 1984. *Classification and Regression Trees*. CRC Press.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. 2022. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8493–8502, Dublin, Ireland. Association for Computational Linguistics.

Feng-Lei Fan, Yingxin Li, Tieyong Zeng, Fei Wang, and Hanchuan Peng. 2025. Towards NeuroAI: introducing neuronal diversity into artificial neural networks. *Med-X*, 3(1):2.

Javier Ferrando, Gabriele Sarti, Arianna Bisazza, and Marta R Costa-Jussà. 2024. A primer on the inner workings of transformer-based language models. *arXiv preprint arXiv:2405.00208*.

Team Gemma, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, and 1 others. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Yang He and Lingao Xiao. 2024. Structured pruning for deep convolutional neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(5):2900–2919.

Takeshi Kojima, Itsuki Okimura, Yusuke Iwasawa, Hitomi Yanaka, and Yutaka Matsuo. 2024. On the multilingual ability of decoder-based pre-trained language models: Finding and controlling language-specific neurons. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6919–6971, Mexico City, Mexico. Association for Computational Linguistics.

Yihong Liu, Runsheng Chen, Lea Hirlimann, Ahmad Dawar Hakimi, Mingyang Wang, Amir Hossein Kargaran, Sascha Rothe, François Yvon, and Hinrich Schütze. 2025. On relation-specific neurons in large language models. *arXiv preprint arXiv:2502.17355*.

Telmo Pires, António Vilarinho Lopes, Yannick Assogba, and Hendra Setiawan. 2023. One wide feedforward is all you need. In *Proceedings of the Eighth Conference on Machine Translation*, pages 1031–1044, Singapore. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1).

Tilman Räuker, Anson Ho, Stephen Casper, and Dylan Hadfield-Menell. 2023. Toward transparent ai: A survey on interpreting the inner structures of deep neural networks. In *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pages 464–483.

Noam Shazeer. 2020. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR.

Ran Song, Shizhu He, Shuting Jiang, Yantuan Xian, Shengxiang Gao, Kang Liu, and Zhengtao Yu. 2024. Does large language model contain task-specific neurons? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7101–7113, Miami, Florida, USA. Association for Computational Linguistics.

Suraj Srinivas and R Venkatesh Babu. 2015. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*.

Felix Stahlberg and Bill Byrne. 2017. Unfolding and shrinking neural machine translation ensembles. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1946–1956, Copenhagen, Denmark. Association for Computational Linguistics.

Sainbayar Sukhbaatar, Edouard Grave, Guillaume Lample, Herve Jegou, and Armand Joulin. 2019. Augmenting self-attention with persistent memory. *arXiv preprint arXiv:1907.01470*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32.

Peng Zhao, Guilherme Rocha, and Bin Yu. 2009. The composite absolute penalties family for grouped and hierarchical variable selection. *The Annals of Statistics*, 37(6A):3468 – 3497.

	Gemma-2B	Gemma-9B	Gemma-27B
Model dimensionality (n)	2,304	3,584	4,608
Number of layers ($\lvert\mathcal{W}_\Theta\rvert$)	26	42	46
Feedforward dimensionality (m)	9,216	14,336	36,864
Number of feedforward parameters	1.656B (64%)	6.474B (70%)	23.441B (86%)
Number of feedforward outgoing weight parameters	552M (21%)	2.158B (23%)	7.814B (29%)

Table 6: Hyper-parameters of the Gemma-2 model family related to the feedforward layers. Note that the feedforward dimensionality is half of what is reported in Table 1 in the Gemma-2 technical report (Gemma et al., 2024) due to a difference in nomenclature.

Task	Method	Gecko	Otter	Bison
GSM8K	Standard fine-tuning	17.82	57.54	67.85
(accuracy in %)	NUCL fine-tuning	19.86	59.06	68.76
SuperGLUE	Standard fine-tuning	75.96	89.45	87.00
(average score in %)	NUCL fine-tuning	77.15	89.66	90.02

Table 7: Absolute scores on GSM8K and SuperGLUE of models in the PaLM-2 model family.

Task	Method	Gemma-2B	Gemma-9B	Gemma-27B
GSM8K	Pre-trained baseline	28.43	75.97	81.58
(accuracy in %)	Standard fine-tuning	44.66	77.79	81.96
	NUCL fine-tuning	45.87	80.52	85.06
MBPP	Pre-trained baseline	30.2	52.4	62.0
(3-shot success rate in %)	Standard fine-tuning	30.6	56.0	64.0
	NUCL fine-tuning	31.6	55.2	65.0

Table 8: Absolute scores on GSM8K and MBPP of models in the Gemma-2 model family.

Dataset		Size
MBPP		1K
GSM8K		7.5K
SuperGLUE	BoolQ	9.4K
	CB	250
	COPA	400
	MultiRC	5.1K
	ReCoRD	101K
	RTE	2.5K
	WiC	6K
	WSC	554

Table 9: Number of examples in the fine-tuning corpora.

A Gemma model sizes

Gemma-2[7] models employ very wide feedforward layers, which account for a substantial portion of the model's parameters. As shown in Table 6, the feedforward dimensionality significantly exceeds the model dimensionality, resulting in 64%-86% of the total parameters residing in the feedforward layer, with 21%-29% specifically within its outgoing weight matrices.

B Absolute evaluation scores

In the main paper we report the improvements of using NUCL ($\alpha > 0$) over standard fine-tuning ($\alpha = 0$). The absolute scores are listed in Table 7 for the PaLM-2 family and in Table 8 for the Gemma-2 family.

C Training hyper-parameters

We train our PaLM-2 models in the JAX (Bradbury et al., 2018) framework PAXML[8] on TPU v4 chips with AdaFactor (Shazeer and Stern, 2018) with a batch size of 32. Dropout rates (0.0 or 0.1) and learning rates (0.0001-0.000001) are tuned for standard fine-tuning on the development sets, and then reused for

[7] Terms of use: https://ai.google.dev/gemma/terms

[8] https://github.com/google/paxml

the NUCL runs. NUCL's α-values range between 10^{-2} and 10^{4} (depending on the model size and batch size) and are tuned in powers of 10. To reduce hallucinations in the PaLM 2 models we append the string "`\n[eod]`" to all training examples and truncate predictions after these tokens.

Our Gemma models are fine-tuned in JAX with a dropout rate of 0.1 on TPU v5e chips (8x8 for 2B models, 16x16 for 9B and 27B models). We use a batch size of 16 and a learning rate of 10^{-6} linearly warmed up over 100 steps.

The sizes of the datasets we used for fine-tuning are listed in Table 9.

D Python implementation

Listing 1 shows the JAX-implementation of our three NUCL loss variants *ent*, *gini*, and *mstd*. We simplify the computation of *mstd* in Eq. 5 as follows to improve numerical stability:

$$
\begin{aligned}
\mathcal{L}_{\text{NUCL-mstd}}(\mathbf{r}) &= -\sqrt{\frac{\sum_{j=1}^{n}\left(r_j - \frac{1}{n}\right)^2}{n}} \\
&= -\sqrt{\frac{1}{n}\sum_{j=1}^{n}(r_j^2 + \frac{1}{n^2} - \frac{2r_j}{n})} \\
&= -\sqrt{\frac{1}{n}(\sum_{j=1}^{n} r_j^2 + \frac{1}{n} - \frac{2}{n}\underbrace{\sum_{j=1}^{n} r_j}_{=1})} \\
&= -\sqrt{\frac{1}{n}(\sum_{j=1}^{n} r_j^2 - \frac{1}{n})}
\end{aligned}
\tag{9}
$$

```python
import jax
import jax.numpy as jnp

def compute_nucl(w_param):
    # w_param is an mxn matrix (m: inner dim., n: model dim.)
    w_abs = jnp.abs(w_param)
    w_z = jnp.sum(w_abs, axis=1)
    w_p = w_abs / jnp.expand_dims(w_z, axis=1)
    r2sum = jnp.sum(jnp.square(w_p), axis=1)

    # NUCL-ent (Eq. 3)
    nucl_ent = jnp.log(w_z) + jnp.sum(jax.scipy.special.entr(w_abs), axis=1) / w_z

    # NUCL-gini (Eq. 4)
    nucl_gini = 1.0 - r2sum

    # NUCL-mstd (Eq. 5)
    dim_z = 1.0 / w_p.shape[1]
    nucl_mstd = -jnp.sqrt(dim_z * (r2sum - dim_z))

    return jnp.sum(nucl_ent), jnp.sum(nucl_gini), jnp.sum(nucl_mstd)
```

Listing 1: Python implementation of NUCL