Empirical Upper Bounds for Unstructured Sparsity in Compute-Efficient Language Modeling

Esha Singh[†,♮,§] **Shane Bergsma**[†] **Nolan Simran Dey**[†] **Joel Hestness**[†]

Gavia Gray[†,‡]

[†]Cerebras Systems
[♮]University of California, San Diego
[‡]`gavia.gray@cerebras.net`
[§]`e3singh@ucsd.edu`

Abstract

Sparsity in deep neural networks promises two improvements in computational efficiency, fewer FLOPs spent both to train the network and to perform inference. We find that both may be quantified best using a compute-efficient scaling law. This tool allows us to compare existing methods to train networks with unstructured sparse regularisation and parametrization. In this setting, it is natural to focus on the proportion of weights in the network whose magnitude is below a given threshold and assume that those weights do not affect the output of the network. However, we may not know where that threshold is, so we aim to separate our analysis from a specific threshold. By evaluating the network sparsity at many possible thresholds we can characterise an empirical upper bound on the advantage of sparsity for pre-training large language models. We test this bound comparing the performance of existing sparse regularization methods to find a 15% reduction in pre-training FLOPs or a 30-40% reduction in inference FLOPs and further identify decoupled proximal methods as a promising direction.

1 Introduction

"Compute efficient" scaling laws constrain the training budget and model size one should choose to obtain a loss on a given dataset in order to minimise the compute spent [Hoffmann et al., 2022]. Scaling law driven development therefore progresses by comparison to this law; for example, if an intervention decreases the validation loss by 1% it is equivalent to spending 7% more FLOPs training the original model [1]. This prompts us to focus on the percentage difference from the expected loss according to the FLOPs we have spent. If a number of the weights in these networks are zero during pre-training then we can spend fewer FLOPs, which implies a potential benefit over the existing scaling law.

Sparse weights in deep neural networks have been well-documented in the field but often the benchmark is an arbitrary trade-off in performance versus sparsity in the final trained model [Gale et al., 2019], *without reference to a scaling law*. The best performing methods in this category typically perform post-training sparsification; removing parameters from the model and adjusting the weights to maintain performance [Frantar and Alistarh, 2023]. While this minimizes inference cost it has no effect on training cost.

[1]7% is an example using the scaling law of Dey et al. [2023].

During conventional training (e.g. GPT2-style [Radford et al., 2019]) weight decay will drive some weights to zero. This existing sparsity is a natural choice for our investigation as it is difficult to predict the FLOPs saved, compared to something like gradual magnitude pruning [Evci et al., 2021] where the pruning schedule (and therefore FLOPs spent) is known in advance. In addition, we may try alternative parametrizations and regularizers to observe their effect. In this paper, we argue that an *upper-bounded* advantage is an effective way to rank these sparsification methods. Specifically, our contributions are as follows:

- We propose empirical upper bound metrics grounded in the compute efficient scaling law that quantify the FLOPs saved at training and inference time versus the chosen scaling law, marginalised over the threshold below which we consider weights dropped. This method is described in Section 5.
- As it is well known that minimizing the L1 of parameters will induce a sparse Laplacian weight distribution [Tibshirani, 1996] we test a family of existing sparsifying regularizers on a 111M language model task to observe the induced scaling law shift. The compared methods are described in Section 4.
- Using our existing scaling laws we extrapolate to characterise the potential benefits from unstructured sparsity when we can assume perfect hardware acceleration and separate existing methods by their potential advantage. These results are found in Section 5.2.

Establishing the upper bound on sparsity's potential for compute-efficiency can help researchers understand the potential return-on-investment for further efforts in the sparsity domain. These efforts have typically included the development of hardware that can accelerate sparsity [Mishra et al., 2021, Cerebras Systems, 2024], software that can better leverage existing hardware [Neural Magic, 2024] and new algorithms that increase sparsity during [Bambhaniya et al., 2024] or after [Frantar and Alistarh, 2023] training.

2 Related work

Prior work in sparse deep learning can be separated into two eras, which we can imprecisely refer to as the "overfitting" and "underfitting" eras. In the "overfitting" era work was focused on datasets that could be processed by the model many times during training, typically image classification datasets such as Imagenet [Krizhevsky et al., 2012, Han et al., 2016, 2015] and CIFAR-10 [Krizhevsky, 2009]; all the earliest work was in this era [LeCun et al., 1989, Hanson and Pratt, 1988, Hassibi and Stork, 1992]. In the "underfitting" era the dataset and model are so large that iterating through each time is expensive so each example might only be seen by the model once [Hestness et al., 2017].

Placed at the boundary between these two eras, Gale et al. [2019] apply prior methods from the overfitting era to the tasks in the overfitting era and find that performance is "inconsistent" at scale. Specifically, they find that a type of magnitude pruning [Zhu and Gupta, 2017] outperformed competing methods on a language modelling task at the time. Since then novel magnitude based pruning methods have been proposed that operate during training [Peste et al., 2021, Saxena et al., 2023, Thangarasa et al., 2023] but have failed to demonstrate Pareto improvement in the scaling law. While work has focused on fitting scaling laws to sparsely trained networks [Frantar et al., 2023] we are not aware of scaling law driven evaluation of sparsity, which we argue here is necessary for useful comparisons.

In this work we focus on how regularisation and parametrization affects sparsity during training, for example L1 regularization is known to induce a Laplace distribution in the weights [Tibshirani, 1996, Hanson and Pratt, 1988, Bengio, 2012, Duchi et al., 2008]. This prompted investigation into the regularization effect of minimizing < 1-norms, such as the L0 norm [Collins and Kohli, 2014, Louizos et al., 2018]. In this work, we focus on regularizers that can be implemented by modifying only the parametrization of the weights and the update function of the optimizer (which is still a very large set) and compare a family of such regularizers in Section 4.

There are also many regularizers that meet these criteria that were not included, such as Deep-Hoyer [Yang et al., 2020], "Polarization" based methods [Zhuang et al., 2020], Continuous Sparsification [Savarese et al., 2021], PowerPropagation Schwarz et al. [2021], Dynamic Sparse Reparametrization Mostafa and Wang [2019]. Our selection criteria focused on what appeared promising and available but we did not exhaustively implement everything that could be implemented. Methods that do not rely on parametrization or update rules were not included despite the popularity of methods such as Rigging the Lottery (RiGL) [Evci et al., 2021].

3 Background

Sparsity in neural networks typically refers to the proportion of weights less than a threshold ϵ. If $\mathcal{W}$ contains all the weights in a network then sparsity s is:

$$s(\epsilon) = \frac{|\{w \in \mathcal{W} : w < \epsilon\}|}{|\mathcal{W}|} \quad \text{and} \quad \rho(\epsilon) = 1 - s(\epsilon) \tag{1}$$

where we have also defined the density ρ to be the proportion of weights greater than ϵ. As mentioned above, we assume that any weights that are ϵ sparse are not computed and we consider these FLOPs saved [2].

If we reach the same loss without using as many FLOPs then we have changed the relationship between FLOPs spent and final loss. Predicting the loss a network may achieve given a FLOP and parameter budget is the domain of neural scaling laws [Kaplan et al., 2020], for example Hoffmann et al. [2022] give the following:

$$\hat{L}(N, D) = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}, \tag{2}$$

where $\hat{L}$ is loss, N is the number of parameters, D is the number of training tokens and E is the irreducible entropy of the dataset[3]. This equation is useful because it produces a prescription for the optimal model size and training budget to obtain a given loss on the dataset the scaling law has been fit. It can also be simplified to produce a scaling law relating the FLOP budget to the validation loss directly,

$$\hat{L}(f) = \left(\frac{f}{f_b}\right)^{-\gamma} + E, \tag{3}$$

where f_b are base FLOPs and γ is another exponent that must be fit to data. In this work we use this scaling law fit to the Pile dataset Gao et al. [2020], so $E = 0.5066$, $f_b = 5.984 \times 10^{22}$ and $\gamma = 0.07037$, as reported by Dey et al. [2023].

4 Sparsifying Regularization

In this work we restrict our investigation to methods whose interaction with the optimization can be precisely characterized by their *parametrization* and *update rules*. Parametrization can be restricted here to only consider functions that transform the stored weights before they are used. Specifically, if the network output y is a function of input x and weights $\mathcal{W}$, the parametrization, Φ, is a function of the weights:

$$y = f(x, \Phi(\mathcal{W}; \mathcal{U})) \quad \text{where} \quad \Phi(\mathcal{W}; \mathcal{U}) = \{\omega \mid \omega = \phi(W, U), W \in \mathcal{W}, U \in \mathcal{U}\} \tag{4}$$

ϕ may or may not have it's own learnable parameters $\mathcal{U}$ and on the right we note that the parametrization Φ is applied to each parameter independently.

In this paper "update rules" refer to two specific functions: PreReg and Update in Algorithm 1. By defining these two functions we can implement various different regularization algorithms. For example, traditional L1 regularization in deep learning is implemented by adding an L1 penalty to the loss:

$$L = L_{\text{task}}(Y, X, \mathcal{W}) + \lambda ||\mathcal{W}||_1 \quad \text{and} \quad \nabla_{\mathcal{W}} L = \nabla_{\mathcal{W}} L_{\text{task}}(Y, X, \mathcal{W}) + \lambda \text{sign}(\mathcal{W}) \tag{5}$$

In our framework, this corresponds to an identity parametrization $\mathcal{W} = \Phi(\mathcal{W})$ and $\text{PreReg}(\mathcal{W}) = \lambda ||\mathcal{W}||_1$. This makes it clear that this method of L1 regularisation is *coupled* [Loshchilov and Hutter, 2019] to the adaptive optimizer, the L1 penalty gradient is passed to the adaptive optimizer state.

[2]To compute the FLOPs saved we can use the standard FLOPs/token $\approx 6 \times$ parameters or account for differing FLOP costs in different layers as described in Appendix A.

[3]For example, fitted parameters provided by Hoffmann et al. [2022, App.D.2] are $E = 1.69$, $A = 406$, $B = 410.7$, $\alpha = 0.34$, $\beta = 0.28$.

Algorithm 1 Generic Optimizer Step

```
Require: $\mathcal{W}_t$ (current weights), $\nabla f(\mathcal{W}_t)$ (gradient), $\mathbf{o}_t$ (optimizer state)
Require: PreReg$(\cdot)$ (pre-update regularizer)
Require: Update$(\cdot)$ (weight update function)
Require: WeightDelta$(\cdot)$ (maybe a adaptive function, but not necessarily)
 1: $\tilde{\nabla} f(\mathcal{W}_t) \leftarrow$ PreReg$(\nabla f(\mathcal{W}_t), \mathcal{W}_t, \mathbf{o}_t)$        ▷ Apply pre-update regularization
 2: $\Delta \mathcal{W}_t \leftarrow$ WeightDelta$(\tilde{\nabla} f(\mathcal{W}_t), \mathcal{W}_t, \mathbf{o}_t)$        ▷ Compute weight change
 3: $\mathcal{W}_{t+1} \leftarrow$ Update$(\mathcal{W}_t, \tilde{\Delta}\mathcal{W}_t, \mathbf{o}_t)$        ▷ Update weights
 4: $\mathbf{o}_{t+1} \leftarrow$ UpdateState$(\mathbf{o}_t, \tilde{\nabla} f(\mathcal{W}_t), \tilde{\Delta}\mathcal{W}_t)$        ▷ Update optimizer state
        return $\mathcal{W}_{t+1}, \mathbf{o}_{t+1}$
```

	Parametrization	PreReg	Update
AdamW	id	id	$\mathcal{W}_t - \eta\Delta\mathcal{W}_t - \eta\lambda\mathcal{W}_t$
L_1	id	PreReg$(\mathcal{W}) = \lambda\|\|\mathcal{W}\|\|_1$	id
L_p	id	id	pWD$(\mathcal{W}_t - \eta\Delta\mathcal{W}_t, \mathcal{W}_t)$
STR	$\mathcal{S}_g(\mathcal{W}, \mathbf{s})$	id	pSTR$(\mathcal{W}_t - \eta\Delta\mathcal{W}_t, \mathcal{W}_t)$
spred	$\mathcal{W} \odot \mathcal{U}$	id	$\mathcal{W}_t - \eta\Delta\mathcal{W}_t - \eta\lambda\mathcal{W}_t$

Table 1: The family of sparsifying regularizers that we compare in this work. id indicates an identity function. η is learning rate and λ is regularisation weight. 'id' refers to Identity matrix (signifying no contribution). The functional forms of pWD, $\mathcal{S}_g, \mathcal{U}$ and pSTR are described in Appendix B.

The question of how for L_p norms with $p < 1$ has been investigated by Outmezguine and Levi [2024]. In this paper we note that their proposed proximal update is easy to implement in the framework described above. Two of the methods we compare have therefore been mentioned, the full list may be found in Table 1.

5 Empirical Upper Bounds for Sparse Scaling

These scaling law experiments replicate of Cerebras-GPT at the 111 million parameter scale [Dey et al., 2023], training the model on the Pile dataset [Gao et al., 2020] until 2.2 billion tokens have been processed. During training we track the absolute value of the histogram of all "effective" weights and use this to estimate the proportion of parameters in each layer that are required. "Effective" here refers to the weight used in the $matmul$, output by the parametrization. This can then be used to compute the sparse FLOPs that would have been required.

5.1 Tyranny of the Scaling Equation

Any intervention, such as sparsity, that reduces the FLOPs used during pre-training will lead to an increase in the validation loss. Inverting equation 3 dictates that, for any increase in loss, one could have trained a smaller compute-efficient model for fewer FLOPs to obtain the same result,

$$\hat{f}(L) = \exp\left(\log(f_b) - \frac{\log(L-E)}{\gamma}\right). \tag{6}$$

For sparse pre-training to be worthwhile, i.e. cost fewer pre-training FLOPs, it needs to cost fewer FLOPs than this. Specifically, we define *Pre-training Advantage* to be,

$$\Delta f = \hat{f}(L) - s(\epsilon, \bar{\rho}), \tag{7}$$

where s is the pre-training FLOPs scaled by the average group sparsity according to the FLOP equation in Appendix A using mean density $\bar{\rho}$.

We refer to this equation as a "tyranny" because any method increasing sparsity can negate it's own FLOP savings by increasing validation loss. For example, a method achieving 50% average sparsity throughout pre-training must not increase loss by more that 7.1%. This is illustrated for a strict $\epsilon = 2^{-13}$ in Figure 1 and we see in this setting that the only runs breaking even are mostly dense during training but win by maintaining loss. However, at $\epsilon = 2^{-8}$ we see benefits to 70% cumulative

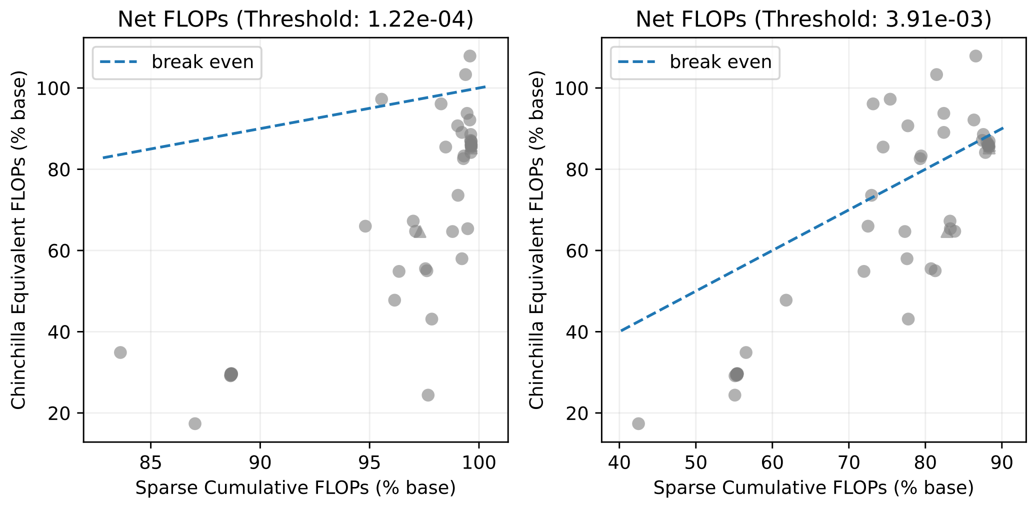

Figure 1: Across all experiments we compute the *Sparse Cumulative FLOPs* for $\epsilon = 2^{-13}/2^{-8}$ (left/right) and compare that to the FLOPs required to train a compute optimal model to the same validation loss, defined as *Chinchilla Equivalent FLOPs*. For sparsity to be worthwhile it's necessary to be above the "break even" line.

sparsity during training; this implies for savings greater than 30% we would need to believe that parameters $> 2^{-8}$ can be set to zero.

A similar relationship holds for sparsity in the final pre-trained model. For any given validation loss we can infer the FLOPs per token, $\hat{f}_d$, of the model size required by the scaling law. In this work, we fit the following expression,

$$\hat{f}_d = 2.6198 \times 10^{12} \times L^{-8.4243}, \tag{8}$$

which can be compared to the final FLOPs per token using the density of the final checkpoint ρ_D,

$$\Delta \hat{f}_d = \hat{f}_d - s_d(\epsilon, \rho_D). \tag{9}$$

where s_d is the sparse FLOPs per token. We refer to this expression as the *Inference Advantage*, quantifying the number of FLOPs saved versus using the equivalent compute-efficient dense model.

5.2 Variable Thresholds

The choice of threshold, ϵ, below which the weight is dropped directly affects the cumulative FLOP estimate. Our analysis produces an *upper bound* because we neglect the performance loss by actually setting the weight to zero. This allows us to compare many different possible threshold settings without retraining the model. The minimum threshold we look at is 2^{-13}, the minimum magnitude that may be expressed in float16, which we take to be a plausible threshold. As the threshold increases the reported upper bound advantage in Figure 2 becomes more loose.

6 Discussion

In this work we neglect initialization, all models are initialized with the same standard deviation as the base model. As shown in Figure 3, sparsity begins very low for all module groups, which reduces the possible FLOP savings accumulated over the entire training run. Sparse initializations are a potential direction to address this [Zhao et al., 2022].

As described in Section 5.2, the absolute thresholds define an upper bound that becomes increasingly loose as the threshold increases. In future work, we hope to define how the loss will be affected by applying a threshold during pre-training and investigate algorithms to apply this threshold. In addition, in post-training quantization it is typical to apply sophisticated algorithms such as SparseGPT [Frantar

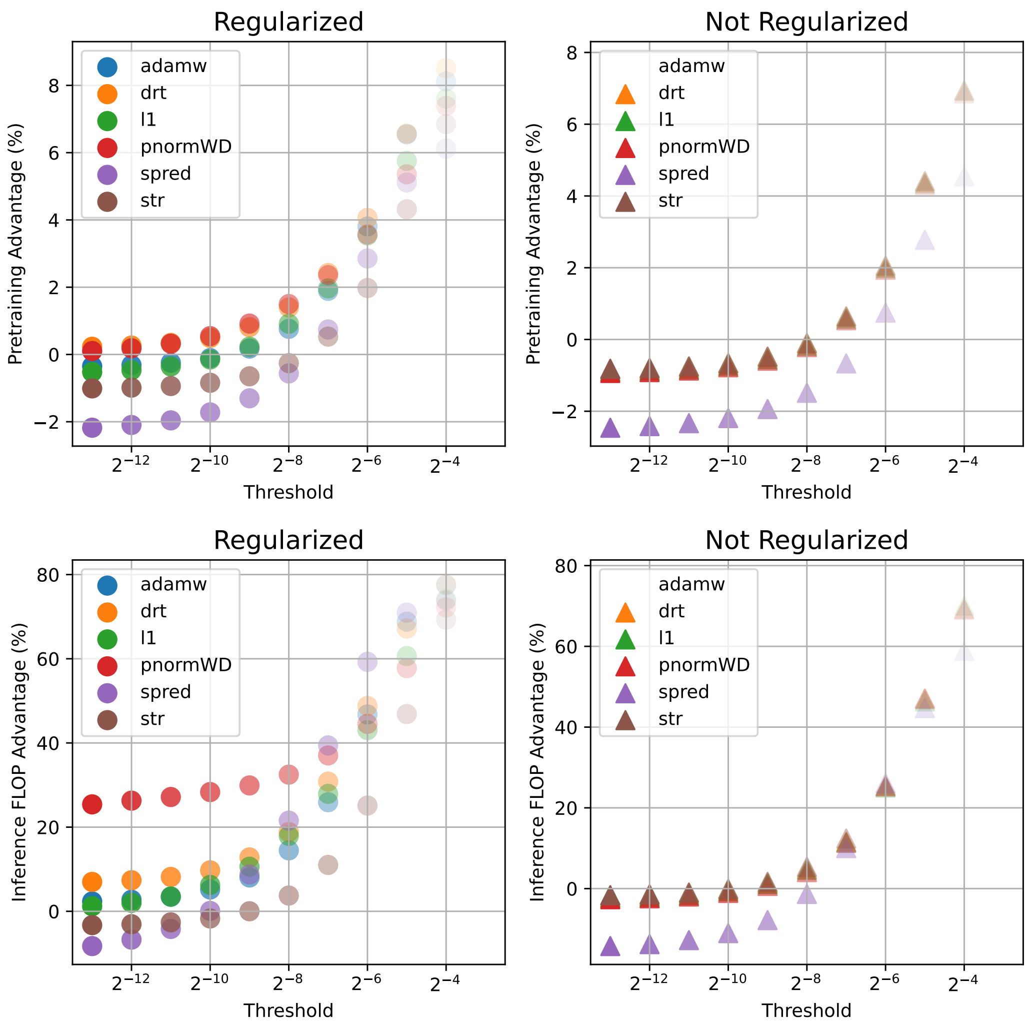

Figure 2: Upper bounds on Pretraining and Inference Advantage as defined in Section 5 are plotted for each of the sparsifying regularizers as described in Section 4. (Left) the best performing hyperparameter config according to each metric is plotted. (Right) the metrics are measured for models that are not regularized for comparison, as a sanity check.

and Alistarh, 2023] and finetune a given model. It is likely that the inherent sparsity in the final weights produced by sparsifying regularization will aid here but we leave the investigation to future work.

7 Conclusion

In this work we have illustrated a way to evaluate sparsity in the domain of large language models, taking compute-efficient scaling laws into account. We define two metrics, pre-training and inference advantage, that allow us to compare a suite of sparsifying regularizers and identify those that may be promising. In doing so we identify an optimal training configuration for sparsifying regularization that promises FLOP savings during pre-training and potentially 30-40% inference advantage.

References

Quentin Anthony, Jacob Hatef, Hailey Schoelkopf, and Stella Biderman. The EleutherAI Model Training Cookbook. GitHub Repo, 2024. URL https://github.com/EleutherAI/cookbook.

Abhimanyu Rajeshkumar Bambhaniya, Amir Yazdanbakhsh, Suvinay Subramanian, Sheng-Chun Kao, Shivani Agrawal, Utku Evci, and Tushar Krishna. Progressive gradient flow for robust N:M sparsity training in transformers. *arXiv preprint arXiv:2402.04744*, 2024.

Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures, 2012. URL https://arxiv.org/abs/1206.5533.

Jeremy Bernstein, Arash Vahdat, Yisong Yue, and Ming-Yu Liu. On the distance between two neural networks and the stability of learning, 2021. URL https://arxiv.org/abs/2002.03432.

Cerebras Systems. Train a model with weight sparsity. Cerebras Systems Documentation, 2024. URL https://docs.cerebras.net/en/2.1.1/wsc/how_to_guides/sparsity.html. Version 2.1.1.

Maxwell D. Collins and Pushmeet Kohli. Memory bounded deep convolutional networks, 2014. URL https://arxiv.org/abs/1412.1442.

Nolan Dey, Gurpreet Gosal, Zhiming, Chen, Hemant Khachane, William Marshall, Ribhu Pathria, Marvin Tom, and Joel Hestness. Cerebras-gpt: Open compute-optimal language models trained on the cerebras wafer-scale cluster, 2023. URL https://arxiv.org/abs/2304.03208.

John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the l1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 272–279, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605582054. doi: 10.1145/1390156.1390191. URL https://doi.org/10.1145/1390156.1390191.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners, 2021. URL https://arxiv.org/abs/1911.11134.

Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337, 2023.

Elias Frantar, Carlos Riquelme, Neil Houlsby, Dan Alistarh, and Utku Evci. Scaling laws for sparsely-connected foundation models, 2023. URL https://arxiv.org/abs/2309.08520.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks, 2019. URL https://arxiv.org/abs/1902.09574.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2020. URL https://arxiv.org/abs/2101.00027.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks, 2015. URL https://arxiv.org/abs/1506.02626.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016. URL https://arxiv.org/abs/1510.00149.

Stephen Hanson and Lorien Pratt. Comparing biases for minimal network construction with back-propagation. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988. URL https://proceedings.neurips.cc/paper_files/paper/1988/file/1c9ac0159c94d8d0cbedc973445af2da-Paper.pdf.

Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992. URL https://proceedings.neurips.cc/paper_files/paper/1992/file/303ed4c69846ab36c2904d3ba8573050-Paper.pdf.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically, 2017. URL https://arxiv.org/abs/1712.00409.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity, 2020. URL https://arxiv.org/abs/2002.03231.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989. URL https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1Y8hhg0b.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization, 2019. URL https://arxiv.org/abs/1902.05967.

Neural Magic. Deepsparse. GitHub repository, 2024. URL https://github.com/neuralmagic/deepsparse.

Nadav Joseph Outmezguine and Noam Levi. Decoupled weight decay for any p norm, 2024. URL https://arxiv.org/abs/2404.10824.

Alexandra Peste, Eugenia Iofinova, Adrian Vladu, and Dan Alistarh. AC/DC: Alternating compressed/decompressed training of deep neural networks, 2021. URL https://arxiv.org/abs/2106.12379.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Pedro Savarese, Hugo Silva, and Michael Maire. Winning the lottery with continuous sparsification, 2021. URL https://arxiv.org/abs/1912.04427.

Shreyas Saxena, Vithursan Thangarasa, Abhay Gupta, and Sean Lie. Sparse iso-FLOP transformations for maximizing training efficiency. In *Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@NeurIPS 2023)*, 2023. URL https://openreview.net/forum?id=iP4WcJ4EX0.

Jonathan Schwarz, Siddhant M. Jayakumar, Razvan Pascanu, Peter E. Latham, and Yee Whye Teh. Powerpropagation: A sparsity inducing weight reparameterisation, 2021. URL https://arxiv.org/abs/2110.00296.

Vithursan Thangarasa, Abhay Gupta, William Marshall, Tianda Li, Kevin Leong, Dennis DeCoste, Sean Lie, and Shreyas Saxena. SPDF: Sparse pre-training and dense fine-tuning for large language models. In Robin J. Evans and Ilya Shpitser, editors, *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence*, volume 216 of *Proceedings of Machine Learning Research*, pages 2134–2146. PMLR, 31 Jul–04 Aug 2023. URL https://proceedings.mlr.press/v216/thangarasa23a.html.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996.

Huanrui Yang, Wei Wen, and Hai Li. Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rylBK34FDS.

Jiawei Zhao, Florian Schäfer, and Anima Anandkumar. Zero initialization: Initializing neural networks with only zeros and ones, 2022. URL https://arxiv.org/abs/2110.12661.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression, 2017. URL https://arxiv.org/abs/1710.01878.

Tao Zhuang, Zhixuan Zhang, Yuheng Huang, Xiaoyi Zeng, Kai Shuang, and Xiang Li. Neuron-level structured pruning using polarization regularizer. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9865–9877. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/703957b6dd9e3a7980e040bee50ded65-Paper.pdf.

Liu Ziyin and Zihao Wang. spred: Solving l_1 penalty with sgd, 2023. URL https://arxiv.org/abs/2210.01212.

A FLOP Calculation

Algorithm 2 describes the method we use to estimate the FLOPs used by the transformers presented in this work. It is based on the work of Anthony et al. [2024]. The dense FLOPs may be computed by summing the elements in TransformerFLOPs and sparse FLOPs may be computed by scaling each element by the average density of that module group.

B Parametrization and Regularization Functions of Regularizers

Below we provide functional formulation for sparsifying regularizers that are discussed in Table 1,

Algorithm 2 Calculate Theoretical FLOPs for Transformer

```
Require:
    $V$: vocabulary size
    $H$: hidden size
    $S$: sequence length
    $L$: number of layers
    $T$: number of tokens ($= S$)
    $I$: iteration factor ($= 3$)
Ensure: TransformerFLOPs object containing:
    $F_{qkv}$: FLOPs for query, key, and value projections
    $F_{am}$: FLOPs for attention matrix computation
    $F_{av}$: FLOPs for attention over values
    $F_{lp}$: FLOPs for linear projection
    $F_{ffn}$: FLOPs for feed-forward network
    $F_{emb}$: FLOPs for token embedding
1: procedure THEORYFLOPS($V, H, S, L$)
2:     $T \leftarrow S$
3:     $I \leftarrow 3$
4:     $F_{qkv} \leftarrow I \cdot 2 \cdot 3 \cdot L \cdot T \cdot H^2$
5:     $F_{am} \leftarrow I \cdot 2 \cdot L \cdot T \cdot S \cdot H$
6:     $F_{av} \leftarrow I \cdot 2 \cdot L \cdot T \cdot S \cdot H$
7:     $F_{lp} \leftarrow I \cdot 2 \cdot L \cdot T \cdot H^2$
8:     $F_{ffn} \leftarrow I \cdot 16 \cdot L \cdot T \cdot H^2$
9:     $F_{emb} \leftarrow 6 \cdot T \cdot H \cdot V$
10:    return TransformerFLOPs($F_{qkv}, F_{am}, F_{av}, F_{lp}, F_{ffn}, F_{emb}$)
11: end procedure
```

- **pWD**: refers to *decoupled weight decay for any p norm* (Outmezguine and Levi [2024]). The functional form of the regularizer is defined as:

$$R_p(w, s) = \frac{\lambda_p}{p} \sum_i [s_i w_i^2 + K(s_i)]$$

$$K(s_i) = \frac{2-p}{p} s_i^{\frac{p}{p-2}}$$

where $p > 0$, $\lambda_p \in R^+$, $||.||_p$ is the p-norm, and $w_i, s_i \in \mathbb{R}, i = 1, ..., N_w$

- **STR**: refers to *Soft-thresholding regularization* (Kusupati et al. [2020]). As defined in Table 1, $\mathcal{S}_g(\mathcal{W}, s)$ is projection of $\mathcal{W}$ parameterized by s (learnable parameter) and function g. $\mathcal{S}$ is applied to each element of $\mathcal{W}$ and is defined as:

$$\mathcal{S}_g(w, s) := sign(w) \cdot \text{ReLU}(|w| - g(s))$$

where $g : \mathbb{R} \longrightarrow \mathbb{R}$, and $\alpha = g(s)$ is the pruning threshold. ReLU$(a) = max(a, 0)$. That is, if $|w| \geq g(s)$, then $S_g(w, s)$ sets it to 0.

- **spred**: refers to method introduced by Ziyin and Wang [2023] called *spred*, where the loss function is parameterized by two learnable parameters V_s, V_d, L_1, regularization strength is 2κ, then optimization problem we solve for is -

$$\min_{W,U,V_d} L(U \odot W, V_d) + \kappa(||W||_2^2 + ||U||^2)$$

such, that $|U_i| = |W_i|$ for all i. In other words, after parameterizing the weights as an element-wise product we can train with normal decoupled weight decay.

- **DRT** (Deep Relative Trust): refers to concept of *"Deep Relative Trust"* introduced in Bernstein et al. [2021]. This sparsfiying regularizer uses the Fromage optimizer's learning rule correction as a regularizer. Since Fromage update is derived from concept of DRT, we name the method as "DRT" regularizer. The motivation to do so is to control the layer-wise perturbations and penalize excessive growth in magnitude through depth. We define the regularizer as -

$$R(W) = \frac{||W_l||_F}{||g||_F}$$

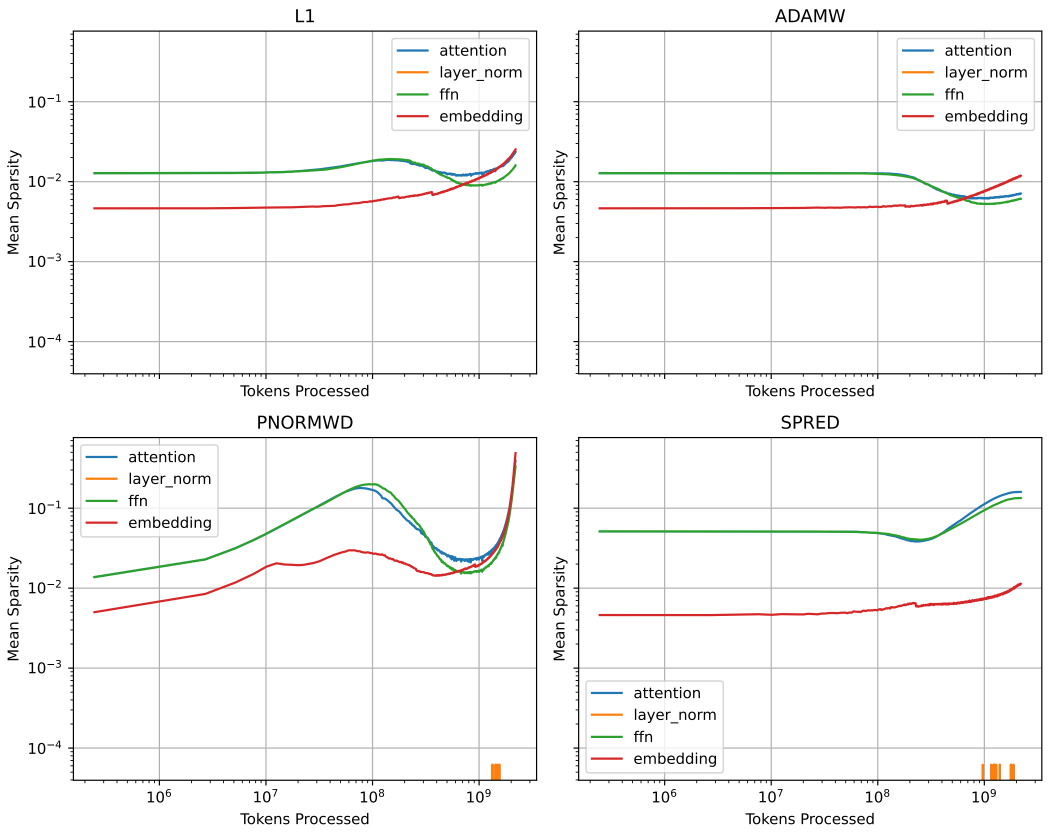

Figure 3: The sparsity in each module group is plotted over the course of training for the config with the best advantage as described in Section 5. Sparsity is defined here as in Section 3 with $\epsilon = 2^{-13}$.

C Sparsity by Module Group

Histograms were gathered every 10 steps for all weight tensors during training, which allows us to plot the progression of sparsity in each weight group over the course of training. Figure 3 illustrates this for the same "best" hyper-parameter settings shown in Figure 2.