RIGGING THE LOTTERY: MAKING ALL TICKETS WINNERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Compared to dense networks, sparse neural networks are shown to be more parameter efficient, more compute efficient and have been used to decrease wall clock inference times. There is a large body of work on training dense networks to yield sparse networks for inference, but this limits the size of the largest trainable sparse model to that of the largest trainable dense model. In this paper we introduce a method to train sparse neural networks with a fixed parameter count and a fixed computational cost throughout training, without sacrificing accuracy relative to existing dense-to-sparse training methods. Our method updates the topology of the sparse network during training by using parameter magnitudes and infrequent gradient calculations. We show that this approach requires fewer floating-point operations (FLOPs) to achieve a given level of accuracy compared to prior techniques. We demonstrate state-of-the-art sparse training results on a variety of networks and datasets, including ResNet-50, MobileNets on Imagenet-2012, and RNNs on WikiText-103.

1 INTRODUCTION

The parameter and floating point operation (FLOP) efficiency of sparse neural networks is now well demonstrated on a variety of problems (Han et al., 2015; Srinivas et al., 2017). Multiple works have shown inference time speedups are possible using sparsity for both Recurrent Neural Networks (RNNs) (Kalchbrenner et al., 2018) and Convolutional Neural Networks (ConvNets) (Park et al., 2016; Elsen et al., 2019). Currently, the most accurate sparse models are obtained with techniques that require, at a minimum, the cost of training a dense model in terms of memory and FLOPs (Zhu & Gupta, 2018; Guo et al., 2016), and sometimes significantly more (Molchanov et al., 2017).

This paradigm has two main limitations. First, the maximum size of sparse models is limited to the largest dense model that can be trained; even if sparse models are more parameter efficient, we can't use pruning to train models that are larger and more accurate than the largest possible dense models. Second, it is inefficient; large amounts of computation must be performed for parameters that are zero valued or that will be zero during inference. Additionally, it remains unknown if the performance of the current best pruning algorithms is an upper bound on the quality of sparse models. Gale et al. (2019) found that three different dense-to-sparse training algorithms all achieve about the same sparsity / accuracy trade-off. However, this is far from conclusive proof that no better performance is possible.

The Lottery Ticket Hypothesis (Frankle & Carbin, 2019) hypothesized that if we can find a sparse neural network with iterative pruning, then we can train that sparse network from scratch, to the same level of accuracy, by *starting from the original initial conditions*. In this paper we introduce a new method for training sparse models without the need of a "lucky" initialization; for this reason, we call our method "The Rigged Lottery" or *RigL*[*]. We make the following specific contributions:

- We introduce *RigL* - an algorithm for training sparse neural networks that requires memory and computational cost proportional to density of the network.
- We perform an extensive empirical evaluation of *RigL* on computer vision tasks. We show that *RigL* achieves higher quality than all previous techniques for a given computational cost.

[*]Pronounced "wriggle".

- We show the surprising result that *RigL* can find more accurate models than the current best dense-to-sparse training algorithms.

2 Related Work

Research on finding sparse neural networks dates back decades, at least to Thimm & Fiesler (1995) who concluded that pruning weights based on magnitude was a simple and powerful technique. Ström (1997) later introduced the idea of retraining the previously pruned network to increase accuracy. Han et al. (2016b) went further and introduced multiple rounds of magnitude pruning and retraining. This is, however, relatively inefficient, requiring ten rounds of retraining when removing 20% of the connections to reach a final sparsity of 90%. To overcome this problem, Narang et al. (2017) introduced gradual pruning, where connections are slowly removed over the course of a single round of training. Zhu & Gupta (2018) refined the technique to minimize the amount of hyper-parameter selection required. Gale et al. (2019) examined magnitude pruning, L_0 Regularization, and Variational Dropout and concluded that they all achieve about the same accuracy versus sparsity trade-off on ResNet-50 and Transformer architectures.

Training techniques that allow for sparsity throughout the entire training process were, to our knowledge, first introduced in Deep Rewiring (DeepR) (Bellec et al., 2018). Sparse Evolutionary Training (SET) (Mocanu et al., 2018) proposed a simpler scheme where weights are pruned according to the standard magnitude criterion used in pruning and are added back at random. Dynamic Sparse Reparameterization (DSR) (Mostafa & Wang, 2019) introduced the idea of allowing the parameter budget to shift between different layers of the model, allowing for non-uniform sparsity. This allows the model to distribute parameters where they are most effective. Unfortunately, the models under consideration are mostly convolutional networks, so the result of this parameter reallocation (which is to decrease the sparsity of early layers and increase the sparsity of later layers) has the overall effect of increasing the FLOP count because the spatial size is largest in the early layers. Sparse Networks from Scratch (SNFS) (Dettmers & Zettlemoyer, 2019) introduces the idea of using the momentum of each parameter as the criterion to be used for growing weights and demonstrates it leads to an improvement in test accuracy. Like DSR, they allow the sparsity of each layer to change and focus on a constant parameter, not FLOP, budget. Importantly, the method requires computing gradients and updating the momentum for *every* parameter in the model, even those that are zero, at *every* iteration. This can result in a significant amount of overall computation. Additionally, depending on the model and training setup, the required storage for the full momentum tensor could be prohibitive. Single-Shot Network Pruning (SNIP) (Lee et al., 2019) attempts to find an initial mask with one-shot pruning and uses the saliency score of parameters to decide which parameters to keep. After pruning, training proceeds with this static sparse network.

There has also been a line of work investigating the Lottery Ticket Hypothesis (Frankle & Carbin, 2019). Frankle et al. (2019) showed that the formulation must be weakened to apply to larger networks such as ResNet-50 (He et al., 2015). In large networks, instead of the original initialization, the values after thousands of optimization steps must be used for initialization. Zhou et al. (2019) showed that "winning lottery tickets" obtain non-random accuracies even before training has started. Though the possibility of training sparse neural networks with a fixed sparsity mask using lottery tickets is intriguing, it remains unclear whether it is possible to generate such initializations – for both masks and parameters – *de novo*.

3 Rigging The Lottery

Our method, *RigL*, is illustrated in Figure 1. *RigL* starts with a random sparse network, and at regularly spaced intervals it removes a fraction of connections based on their magnitudes and activates new ones using instantaneous gradient information. After updating the connectivity, training continues with the updated network until the next update. The main parts of our algorithm are explained below.

(0) Notation. Given a dataset D with individual samples x_i and targets y_i, we aim to minimize the loss function $\sum_i L(f_\Theta(x_i), y_i)$, where $f_\Theta(\cdot)$ is a neural network with parameters $\Theta \in \mathbb{R}^N$. Parameters of the l^{th} layer are denoted with Θ^l which is a length N^l vector. A sparse layer keeps only a fraction $s^l \in (0, 1)$ of its connections and parameterized with vector θ^l of length $(1 - s^l)N^l$.

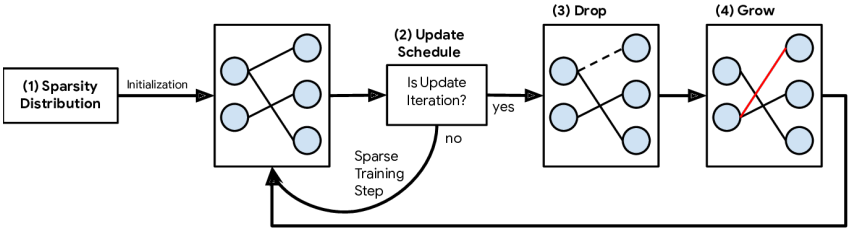

Figure 1: *RigL* improves the optimization of sparse neural networks by leveraging weight magnitude and gradient information to jointly optimize model parameters and connectivity.

Parameters of the corresponding sparse network is denoted with θ. Finally, the overall sparsity of a sparse network is defined as the ratio of zeros to the total parameter count, i.e. $S = \frac{\sum_l s^l N^l}{N}$

(1) Sparsity Distribution. There are many ways of distributing the non-zero weights across the layers while maintaining a certain overall sparsity. We avoid re-allocating parameters between layers during the training process as it makes it difficult to target a specific final FLOP budget, which is important for many applications. We consider the following strategies:

1. *Uniform:* The sparsity s^l of each individual layer is equal to the total sparsity S. In this setting, we keep the first layer dense, since sparsifying this layer has a disproportional effect on the performance and almost no effect on the total size.
2. *Erdős-Rényi-Kernel (ERK):* This method modifies the original Erdős-Rényi Mocanu et al. (2018) formulation by including the kernel dimensions in the scaling factors. In other words, the number of parameters of the sparse convolutional layers are scaled proportional to $1-\frac{n^{l-1}+n^l+w^l+h^l}{n^{l-1}*n^l*w^l*h^l}$, where w^l and h^l are the width and the height of the l'th convolutional kernel. ERK allocates higher sparsities to the layers with more parameters while allocating lower sparsities to the smaller ones.

In all methods, the bias and batch-norm parameters are kept dense, since these parameters scale with total number of neurons and have a negligible effect on the total model size.

(2) Update Schedule. The update schedule is defined by the following parameters: (1)ΔT: the number of iterations between sparse connectivity updates, (2) T_{end}: the iteration at which to stop updating the sparse connectivity, (3) α: the initial fraction of connections updated and (4) f_{decay}: a function, invoked every ΔT iterations until T_{end}, possibly decaying the fraction of updated connections over time. For the latter, as in Dettmers & Zettlemoyer (2019), we use *cosine* annealing, as we find it slightly outperforms the other methods considered.

$$f_{decay}(t;\ \alpha,\ T_{end}) = \frac{\alpha}{2}\left(1 + cos\left(\frac{t\pi}{T_{end}}\right)\right)$$

(3) Drop criterion. Every ΔT steps we drop the connections given by $ArgTopK(-|\theta^l|, (1 - s^l)N^l)$, where $ArgTopK(v,\ k)$ gives the indices of the top-k elements of vector v.

(4) Grow criterion. The novelty of our method lies in how we grow new connections. We grow the connections with highest magnitude gradients, $ArgTopK_{i \notin \theta^l \setminus \mathbb{I}_{drop}}(|\nabla_{\Theta^l} L|,\ k)$, where $\theta^l \setminus \mathbb{I}_{drop}$ is the set of active connections remaining after step (3). Newly activated connections are initialized to zero and therefore don't affect the output of the network. However they are expected to receive gradients with high magnitudes in the next iteration and therefore reduce the loss fastest. As long as $\Delta T > \frac{1}{1-s}$, the extra work of calculating dense gradients is amortized and still proportional to $1-S$. This is in contrast to the method of Dettmers & Zettlemoyer (2019), which requires calculating and storing the full gradients at each optimization step.

4 Empirical Evaluation

Our experiments include image classification using CNNs on the ImageNet-2012 (Russakovsky et al., 2015). We observe similar results on language modelling tasks. We repeat all of our experi-

Method	Top-1 Acc.	FLOPs (Train)	FLOPs (Test)	Top-1 Acc.	FLOPs (Train)	FLOPs (Test)
Dense	76.8	1x (3e18)	1x (8e9)			
	S=0.8			S=0.9		
Static	70.6	0.23x	0.23x	65.8	0.10x	0.10x
SNIP	72.0	0.23x	0.23x	67.2	0.10x	0.10x
Small-Dense	72.1	0.20x	0.20x	68.9	0.12x	0.12x
SET	72.9	0.23x	0.23x	69.6	0.10x	0.10x
RigL	74.6	0.23x	0.23x	72.0	0.10x	0.10x
Small-Dense$_{5\times}$	73.9	1.01x	0.20x	71.3	0.60x	0.12x
RigL$_{5\times}$	**76.6**	1.14x	0.23x	**75.7**	0.52x	0.10x
Static (ERK)	72.1	0.42x	0.42x	67.7	0.24x	0.24x
DSR*	73.3	0.40x	0.40x	71.6	0.30x	0.30x
RigL (ERK)	75.1	0.42x	0.42x	73.0	0.25x	0.24x
RigL$_{5\times}$ (ERK)	**77.1**	2.09x	0.42x	**76.4**	1.23x	0.24x
SNFS*	74.2	n/a	n/a	72.3	n/a	n/a
SNFS (ERK)	75.2	0.61x	0.42x	72.9	0.50x	0.24x
Pruning*	75.6	1.00x	0.23x	73.9	1.00x	0.10x
Pruning$_{1.5\times}$*	**76.5**	1.50x	0.23x	**75.2**	1.50x	0.10x

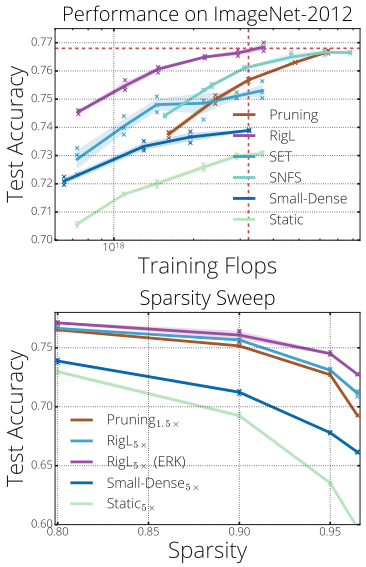

Figure 2: **(left)** Performance and cost of training 80% and 90% sparse ResNet-50s on the Imagenet-2012 classification task. We report FLOPs needed for training and test (inference on single sample) and normalize them with the FLOPs of a dense model (see Appendix A for details on how FLOPs are calculated). Methods with superscript '*' indicates reported results in corresponding papers. Pruning results are obtained from Gale et al. (2019). **(top-right)** Performance of sparse training methods on training 80% sparse ResNet-50 with uniform sparsity distribution. Points at each curve correspond to the individual training runs with training multipliers from 1 to 5 (except pruning which is scaled between 0.5 and 2). The number of FLOPs required to train a standard dense ResNet-50 along with its performance is indicated with a dashed red line. **(bottom-right)** Performance of *RigL* at different sparsity levels with extended training.

ments 3 times and report the mean and standard deviation. We use the TensorFlow Model Pruning library (Zhu & Gupta, 2018) for our pruning baselines. A Tensorflow (Abadi et al., 2015) implementation of our method along with three other baselines (SET, SNFS, SNIP) can be found here[†].

The default number of training steps used for training dense networks might not be optimal for sparse training with dynamic connectivity. In our experiments we observe that sparse training methods benefit significantly from increased training steps. When increasing the training steps by a factor M, the anchor epochs of the learning rate schedule and the end iteration of the mask update schedule are also scaled by the same factor; we indicate this scaling with a subscript (e.g. RigL$_{M\times}$).

In all experiments in this section, we use SGD with momentum as our optimizer. We set the momentum coefficient of the optimizer to 0.9, L_2 regularization coefficient to 0.0001, and label smoothing (Szegedy et al., 2016) to 0.1. The learning rate schedule starts with a linear warm up reaching its maximum value of 1.6 at epoch 5 which is then dropped by a factor of 10 at epochs 30, 70 and 90. We train our networks with a batch size of 4096 for 32000 steps which roughly corresponds to 100 epochs of training. Our training pipeline uses standard data augmentation, which includes random flips and crops.

4.1 ResNet-50 on ImageNet-2012 Task

Figure 2-top-right summarizes the performance of various methods on training an 80% sparse ResNet-50. We also train small dense networks with equivalent parameter count. All sparse networks use a uniform layer-wise sparsity distribution unless otherwise specified and a cosine update schedule ($\alpha = 0.3$, $\Delta T = 100$). Overall, we observe that the performance of all methods improves with training time; thus, for each method we run extended training with up to 5$\times$ the training steps of the original.

[†]https://bit.ly/icml_rigl

As noted by Gale et al. (2019), Evci et al. (2019), Frankle et al. (2019), and Mostafa & Wang (2019), training a network with fixed sparsity from scratch (*Static*) leads to inferior performance. Training a dense network with the same number of parameters (*Small-Dense*) gets better results than *Static*, but fails to match the performance of dynamic sparse models. SET improves the performance over *Small-Dense*, however saturates around 75% accuracy indicating the limits of growing new connections randomly. Methods that use gradient information to grow new connections (*RigL* and SNFS) obtain higher accuracies, but *RigL* achieves the highest accuracy and does so while consistently requiring fewer FLOPs than the other methods.

Given that different applications or scenarios might require a limit on the number of FLOPs for inference, we investigate the performance of our method at various sparsity levels. As mentioned previously, one strength of our method is that its resource requirements are constant throughout training and we can choose the level of sparsity that fits our training and/or inference constraints. In Figure 2-bottom-right we show the performance of our method at different sparsities and compare them with the pruning results of Gale et al. (2019), which uses 1.5x training steps, relative to the original 32k iterations. To make a fair comparison with regards to FLOPs, we scale the learning schedule of all other methods by 5x. Note that even after extending the training, it takes less FLOPs to train sparse networks using *RigL* compared to the pruning method[‡].

RigL, our method with constant sparsity distribution, **exceeds** the performance of magnitude based iterative pruning in all sparsity levels while requiring less FLOPs to train. Sparse networks that use *Erdős-Renyi-Kernel (ERK)* sparsity distribution obtains even greater performance. For example ResNet-50 with 96.5% sparsity achieves a remarkable 72.75% Top-1 Accuracy, around 3.5% higher than the extended magnitude pruning results reported by Gale et al. (2019). As observed earlier, smaller dense models (with the same number of parameters) or sparse models with a static connectivity can not perform at a comparable level.

A more fine grained comparison of sparse training methods is presented in Figure 2-left. Methods using uniform sparsity distribution and whose FLOP/memory footprint scales directly with (1-S) are placed in the first sub-group of the table. The second sub-group includes DSR and networks with ERK sparsity distribution which require a higher number of FLOPs for inference with same parameter count. The final sub-group includes methods that require the space and the work proportional to training a dense model.

5 Discussion & Conclusion

In this work we introduced 'Rigged Lottery' or *RigL*, an algorithm for training sparse neural networks efficiently. For a given computational budget *RigL* achieves higher accuracies than existing dense-to-sparse and sparse-to-sparse training algorithms. *RigL* is useful in three different scenarios: (1) To improve the accuracy of sparse models intended for deployment; (2) To improve the accuracy of large sparse models which can only be trained for a limited number of iterations; and (3) Combined with sparse primitives to enable training of extremely large sparse models which otherwise would not be possible.

Progress in the first two scenarios are important for adapting state of art Machine Learning algorithms and models in resource constrained settings and *RigL* is pushing the frontier. The third scenario is unexplored due to the lack of hardware and software support for sparsity. Nonetheless, work continues to improve the performance of sparse networks on current hardware (Hong et al., 2019; Merrill & Garland, 2016), and new types of hardware accelerators will have better support for parameter sparsity (Wang et al., 2018; Mike Ashby, 2019; Liu et al., 2018; Han et al., 2016a; Chen et al., 2019). *RigL* provides the tools to take advantage of, and motivation for, such advances.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah,

[‡]Except for the 80% sparse *RigL*-ERK

Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/.

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert A. Legenstein. Deep rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*, 2018.

Y. Chen, T. Yang, J. Emer, and V. Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, June 2019. doi: 10.1109/JETCAS.2019.2910232.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *ArXiv*, 2019. URL http://arxiv.org/abs/1907.04840.

Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets. *ArXiv*, 2019. URL https://arxiv.org/abs/1911.09723.

Utku Evci, Fabian Pedregosa, Aidan N. Gomez, and Erich Elsen. The difficulty of training sparse neural networks. *ArXiv*, 2019. URL http://arxiv.org/abs/1906.10732.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. The lottery ticket hypothesis at scale. *ArXiv*, 2019. URL http://arxiv.org/abs/1903.01611.

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

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient DNNs. *CoRR*, abs/1608.04493, 2016. URL http://arxiv.org/abs/1608.04493.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, 2015.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. EIE: Efficient Inference Engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*, 2016a.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016b. URL http://arxiv.org/abs/1510.00149.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

Changwan Hong, Aravind Sukumaran-Rajam, Israt Nisa, Kunal Singh, and P. Sadayappan. Adaptive sparse tiling for sparse matrix multiplication. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, PPoPP '19, pp. 300–314, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6225-2. doi: 10.1145/3293883.3295712. URL http://doi.acm.org/10.1145/3293883.3295712.

Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient neural audio synthesis. In *International Conference on Machine Learning (ICML)*, 2018.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. SNIP: Single-shot Network Pruning based on Connection Sensitivity. In *International Conference on Learning Representations (ICLR), 2019*, 2019.

Chen Liu, Guillaume Bellec, Bernhard Vogginger, David Kappel, Johannes Partzsch, Felix Neumaerker, Sebastian Höppner, Wolfgang Maass, Stephen B. Furber, Robert A. Legenstein, and Christian Mayr. Memory-efficient deep learning on a spinnaker 2 prototype. In *Front. Neurosci.*, 2018.

Duane Merrill and Michael Garland. Merge-based sparse matrix-vector multiplication (spmv) using the csr storage format. In *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '16, pp. 43:1–43:2, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4092-2. doi: 10.1145/2851141.2851190. URL http://doi.acm.org/10.1145/2851141.2851190.

Peter Baldwin Martijn Bastiaan Oliver Bunting Aiken Cairncross Christopher Chalmers Liz Corrigan Sam Davis Nathan van Doorn Jon Fowler Graham Hazel Basile Henry David Page Jonny Shipton Shaun Steenkamp Mike Ashby, Christiaan Baaij. Exploiting unstructured sparsity on next-generation datacenter hardware. 2019. URL https://myrtle.ai/wp-content/uploads/2019/06/IEEEformatMyrtle.ai_.21.06.19_b.pdf.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 2018.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry P. Vetrov. Variational Dropout Sparsifies Deep Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 2498–2507, 2017.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pp. 4646–4655, 2019. URL http://proceedings.mlr.press/v97/mostafa19a.html.

Sharan Narang, Greg Diamos, Shubho Sengupta, and Erich Elsen. Exploring sparsity in recurrent neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. URL https://openreview.net/forum?id=BylSPv9gx.

Jongsoo Park, Sheng R. Li, Wei Wen, Hai Li, Yiran Chen, and Pradeep Dubey. Holistic SparseCNN: Forging the trident of accuracy, speed, and size. *CoRR*, abs/1608.01409, 2016. URL http://arxiv.org/abs/1608.01409.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 2015.

S. Srinivas, A. Subramanya, and R. V. Babu. Training sparse neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 455–462, July 2017. doi: 10.1109/CVPRW.2017.61.

Nikko Ström. Sparse Connection and Pruning in Large Dynamic Artificial Neural Networks. In *EUROSPEECH*, 1997.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition,*, 2016. URL http://arxiv.org/abs/1512.00567.

Georg Thimm and Emile Fiesler. Evaluating pruning methods. In *National Chiao-Tung University*, pp. 2, 1995.

Peiqi Wang, Yu Ji, Chi Hong, Yongqiang Lyu, Dongsheng Wang, and Yuan Xie. Snrram: An efficient sparse neural network computation architecture based on resistive random-access memory. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, pp. 106:1–106:6, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5700-5. doi: 10.1145/3195970.3196116. URL http://doi.acm.org/10.1145/3195970.3196116.

Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing Lottery Tickets: Zeros, Signs, and the Supermask. *ArXiv*, 2019.

Michael Zhu and Suyog Gupta. To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression. In *International Conference on Learning Representations Workshop*, 2018.

A CALCULATING FLOPS OF MODELS AND METHODS

In order to calculate FLOPs needed for a single forward pass of a sparse model, we count the total number of multiplications and additions layer by layer for a given layer sparsity s^l. The total FLOPs is then obtained by summing up all of these multiply and adds.

Different sparsity distributions require different number of FLOPs to compute a single prediction. For example *Erdős-Renyi-Kernel* distributions usually cause earlier layers to be less sparse than the later layers (see Appendix B). The inputs of earlier layers have greater spatial dimensions, so a convolutional kernel that works on such inputs will require more FLOPs to compute the output features compared to later layers. Thus, having earlier layers which are less sparse results in a higher total number of FLOPs required by a model.

Training a neural network consists of 2 main steps:

1. *forward pass:* Calculating the loss of the current set of parameters on a given batch of data. During this process layer activations are calculated in sequence using the previous activations and the parameters of the layer. Activation of layers are stored in memory for the backward pass.
2. *backward pass:* Using the loss value as the initial error signal, we back-propagate the error signal while calculating the gradient of parameters. During the backward pass each layer calculates 2 quantities: the gradient of the activations of the previous layer and the gradient of its parameters. Therefore in our calculations we count backward passes as two times the computational expense of the forward pass. We omit the FLOPs needed for batch normalization and cross entropy.

Dynamic sparse training methods require some extra FLOPs to update the connectivity of the neural network. We omit FLOPs needed for dropping the lowest magnitude connections in our calculations. For a given dense architecture with FLOPs f_D and a sparse version with FLOPs f_S, the total FLOPs required to calculate the gradient on a single sample is computed as follows:

- **Static Sparse and Dense.** Scales with $3 * f_S$ and $3 * f_D$ FLOPs, respectively.
- **Snip.** We omit the initial dense gradient calculation since it is negligible, which means Snip scales in the same way as Static methods: $3 * f_S$ FLOPs.
- **SET.** We omit the extra FLOPs needed for growing random connections, since this operation can be done on chip efficiently. Therefore, the total FLOPs for SET scales with $3 * f_S$.
- **SNFS.** Forward pass and back-propagating the error signal needs $2 * f_S$ FLOPs. However, the dense gradient needs to be calculated at every iteration. Thus, the total number of FLOPs scales with $2 * f_S + f_D$.
- **RigL.** Iterations with no connection updates need $3 * f_S$ FLOPs. However, at every ΔT iteration we need to calculate the dense gradients. This results in the average FLOPs for *RigL* given by $\frac{(3*f_S*\Delta T+2*f_S+f_D)}{(\Delta T+1)}$.

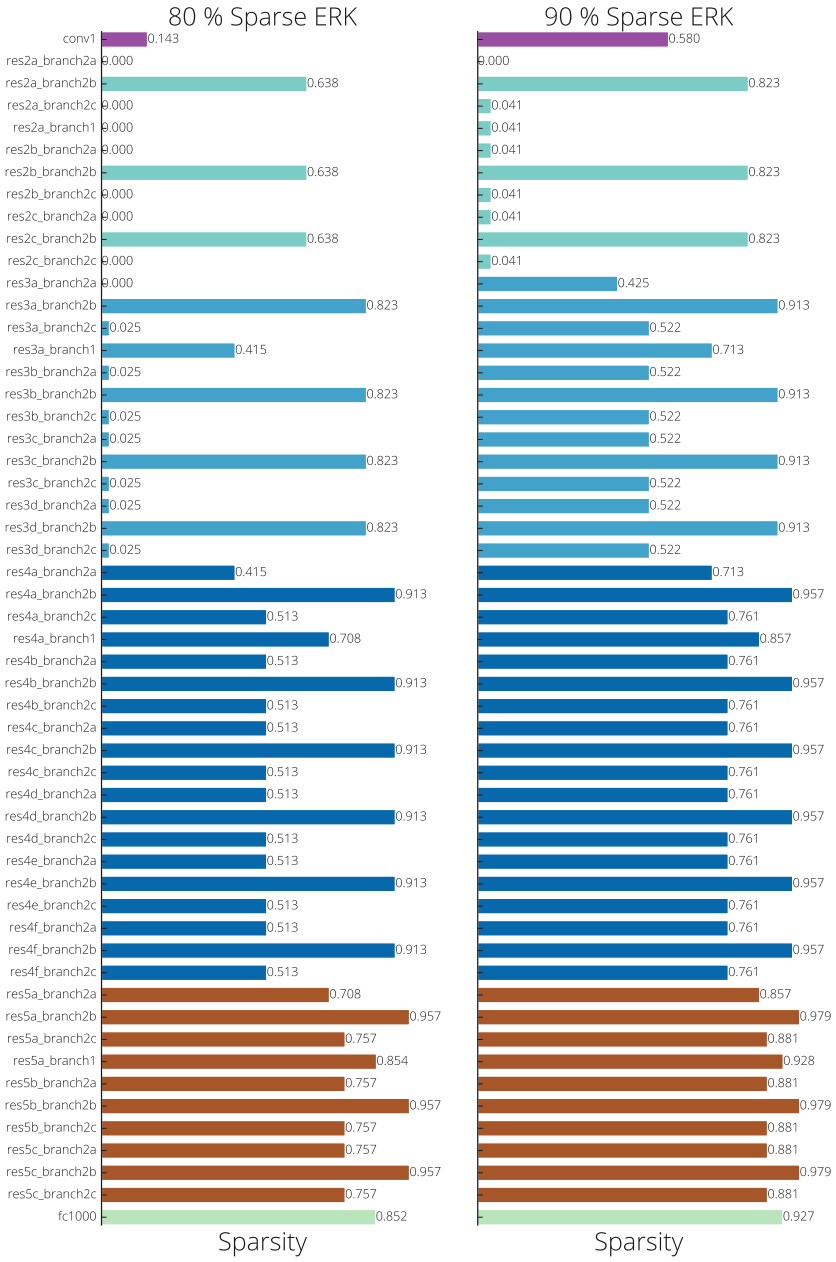

Figure 3: Sparsities of individual layers of the ResNet-50.

B SPARSITY OF INDIVIDUAL LAYERS FOR SPARSE RESNET-50

Sparsity of ResNet-50 layers given by the Erdős-Rényi-Kernel sparsity distribution plotted in Figure 3.