[Supplementary]
Reproducibility Report
Rigging the Lottery: Making All Tickets Winners

Anonymous Author(s)
Affiliation
Address
`email`

1 Architecture Specific Details—ResNet-50 on CIFAR100

Table 1: **ResNet-50 architecture used on CIFAR100**. Building blocks are shown in brackets, with the numbers of blocks stacked. Downsampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

Layer Name	Output Size	ResNet-50
conv1	32×32	3×3, 64, no stride
conv2_x	32×32	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	16×16	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$
conv4_x	8×8	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$
conv5_x	4×4	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 100-d fc, softmax
FLOPs		2.59e9

We use a variant of the originally proposed ResNet architecture (He et al. [2016]). Particularly, we replace the initial 7×7 conv layer with a 3×3 conv layer. Here, "conv layer" refers to convolution followed by batchnorm (Ioffe and Szegedy [2015]) and ReLU activation. This is intended to not excessively downsample the image—CIFAR-100 (Alex Krizhevsky [2009]) has images of dimensions 32×32, compared to Imagenet's (Russakovsky et al. [2015]) 224×224. Each block used (conv2_x, conv3_x, etc.) is a bottleneck block, and uses the conv-batchnorm-ReLU ordering.

2 FLOP Counting Procedure

Following Evci et al. [2020], we base our counting procedure on the Micronet Challenge[1], which was conducted as a part of NeurIPS 2019. Support for unstructured sparsity is assumed while computing the number of additions and multiplication operations. The sum of these two gives us the theoretical FLOPs for a single forward pass through the model.

Concretely, let the FLOPs required for a forward pass through a dense model be f_d and the corresponding for a sparse model (or small-dense model) be f_s. Then, the FLOPs for training a dense model are $3f_d$—since the backward pass involves computing gradients with respect to each weight and activation. f_s can be computed for a model given its sparsity distribution via the counting procedure. The FLOPs required to train a sparse model depend on the technique used, as detailed below.

2.1 Inference FLOPs

Small-Dense, RigL, SET, Static These methods involve constant layer-wise sparsity throughout training, hence the FLOP count can be determined during any step. The FLOP count for Random initialized models are $(1-s)$ times the Dense FLOPs.

SNFS, Pruning Both methods involve varying layer-wise sparsity during training, and hence non-constant FLOP consumption. The final weights are used to determine inference FLOPs in this case.

2.2 Train FLOPs

Small-Dense, Static Dense gradients are not required by these models, and hence have a train FLOP count of $3f_s$.

SET Dense gradients are not required, and random growth can be implemented quite efficiently. Thus, the train FLOP count is $3f_s$.

RigL Dense gradients are required only every ΔT steps, hence the corresponding train FLOP count is: $\frac{3\Delta T f_s + 2f_s + f_d}{\Delta T + 1}$. We note that since ΔT is typically set between 100–1000, the preceding expression is quite close to $3f_s$.

SNFS Dense gradients are required at each training step, resulting in $2f_s + f_d$ FLOPs consumed at each step. Since the sparse FLOP count varies as we train, the average FLOP count is: $2\mathbb{E}[f_{s,t}] + f_d$, where $f_{s,t}$ is the sparse inference FLOPs at train step t.

Pruning Does not require dense gradients, but the sparsity increases smoothly from 0% to the target value as we train. The FLOP consumption here is $3\mathbb{E}[f_{s,t}]$, , where $f_{s,t}$ is the sparse inference FLOPs at train step t.

To determine $\mathbb{E}[f_{s,t}]$, we compute a running average of the FLOP consumption after every epoch. Notably, we find that the inference cost of Pruning is often close to a Random initialized sparse network, while SNFS, regardless of initiazation, is compute-intensive.

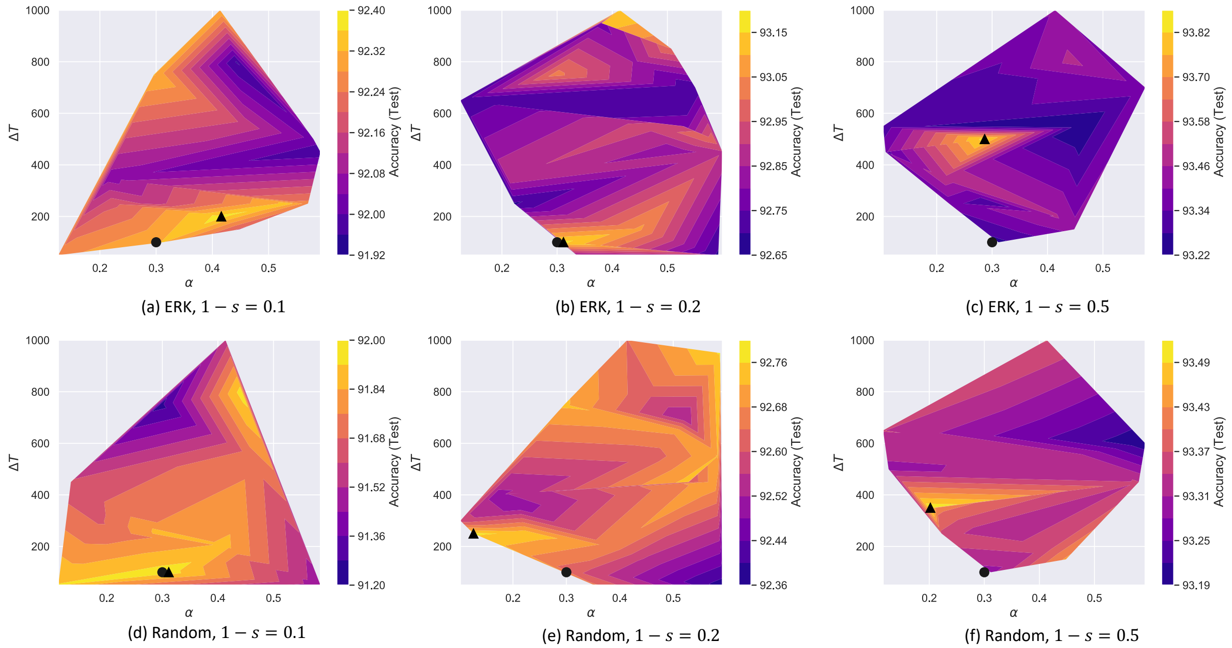
3 Trial Space of Hyperparameter Tuning

Figure 1 shows the hyper-parameter study for tuning $(\alpha, \Delta T)$ as a contour plot. We observe that for multiple initialization-density configurations, the reference choice ($\alpha = 0.3, \Delta T = 100$), is quite close to the optimal hyper-parameters. Furthermore, where they differ, the difference is within standard deviation bounds (Table 4 of the main report).

4 Dynamic Structured Sparsity

Present hardware accelerators lack efficient implementations for unstructured sparsity. As a result, in practice, the reduced FLOP requirement of sparse methods rarely translate to wall-clock improvements. In comparison, there are efficient implementations available for structured (or block) sparsity which reach theoretical speedups (Gray et al. [2017], Teja Vooturi et al. [2019]). Motivated by this, we try modifying *RigL* to explicitly work on structured sparsity.

[1] https://micronet-challenge.github.io/



(a) ERK, $1-s=0.1$ (b) ERK, $1-s=0.2$ (c) ERK, $1-s=0.5$

(d) Random, $1-s=0.1$ (e) Random, $1-s=0.2$ (f) Random, $1-s=0.5$

Figure 1: **Trial space of tuning** $(\alpha, \Delta T)$, shown as a countor plot. Here, black circle corresponds to $(\alpha = 0.3, \Delta T = 100)$, while black triangle corresponds to the optimal hyper-parameter pair found. We plot the convex hull of the trial space, so in a few cases the reference point lies on the border of this space.

Table 2: **Modifying *RigL* for structured sparsity, compared on CIFAR-10 and CIFAR-100 datasets.** *RigL*-struct fails to match the accuracy of *RigL* and just matches Small-Dense in performance.

Method	CIFAR-10				CIFAR-100			
	$1-s=0.1$		$1-s=0.2$		$1-s=0.1$		$1-s=0.2$	
	Accuracy ↑ (Test)	Wall Time ↓	Accuracy ↑ (Test)	Wall Time ↓	Accuracy ↑ (Test)	Wall Time ↓	Accuracy ↑ (Test)	Wall Time ↓
Small-Dense	89.0 ± 0.35	0.11x	91.0 ± 0.07	0.20x	70.8 ± 0.22	0.11x	72.6± 0.93	0.20x
Random Initialization								
RigL	91.7 ± 0.18	1.0x	92.9 ± 0.10	1.0x	71.8 ± 0.33	1.0x	73.5 ± 0.04	1.0x
RigL-Struct	87.0 ± 0.09	0.10x	90.4 ± 0.27	0.20x	69.1 ± 0.11	0.10x	71.9 ± 0.13	0.20x
ERK Initialization								
RigL	92.4 ± 0.06	1.0x	93.1 ± 0.09	1.0x	72.6 ± 0.37	1.0x	73.4 ± 0.15	1.0x
RigL-Struct	89.6 ± 0.16	0.17x	91.3 ± 0.18	0.35x	71.1 ± 0.15	0.23x	72.9 ± 0.08	0.38x

We promote channel sparsity for convolutional layers and keep fully connected layers dense. Mask update steps also operate at the channel level, based on *RigL*'s growth and pruning criterion. We name this method as *RigL*-struct. Such an approach is enticing, as we can remove masked-out channels, and obtain practical speedups on accelerators without needing support for unstructured sparsity.

Unfortunately, *RigL*-struct does not preserve the performance of originally proposed *RigL* (Table 2). In fact, it performs only as good as Small-Dense models, which negates the motivation behind such an experiment—Small-Dense models already achieve the intended speedups.

References

Geoffrey Hinton Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *Proceedings of Machine Learning and Systems (ICML)*, July 2020.

Scott Gray, Alec Radford, and Diederik P Kingma. Gpu kernels for block-sparse weights. *arXiv preprint arXiv:1711.09224*, 3, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, July 2015.

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)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Dharma Teja Vooturi, Girish Varma, and Kishore Kothapalli. Dynamic block sparse reparameterization of convolutional neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, Oct 2019.