LEARNING SPARSE DNNS WITH SOFT THRESHOLDING OF WEIGHTS DURING TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper proposes a new and simple way of training sparse neural networks. Our method is based on a differentiation of the forward and backward paths: the weights in the forward path are a thresholded version of the weights maintained in the backward path. This decoupling allows for micro-updates, produced by gradient descent, to stack up, leading to the possible re-activation of weights that were set to zero in earlier training steps. At the end of training, links with zero weights are pruned away.

Additional critical specificities of our approach lie (i) in the progressive increase of the zeroed weight ratio along the training, and (ii) in the use of soft-thresholding rather than hard-tresholding to derive the forward-path weights from the ones maintained in the backward path. At constant accuracy, our approach reduces the number of training cycles to 1 compared to the state-of-the-art recursive pruning methods. At high pruning rates, it also improves the model accuracy compared to other single cycle pruning approaches (66.18% top-1 accuracy when training a ResNet-50 on ImageNet at 98% sparsity).

1 INTRODUCTION

State-of-the-art neural networks are composed of a few million parameters, leading to a few billion computations per inference. To limit these amounts, sparse networks have been thoroughly investigated in the past few years (Frankle & Carbin (2019); Liu et al. (2019); Lee et al. (2019); Evci et al. (2019); Frankle et al. (2020); Renda et al. (2020)). Those networks reduce the inference complexity by setting most of their weight parameters to zero. Recent works have shown that this can be done without penalizing the prediction accuracy compared to non-sparse networks (cite).

Training methods leading to sparse networks generally build on multiple rounds, or cycles, of iterative gradient descent updates. As detailed in Section 2, each round applies recursively to the outcome of the previous round, after having pruned a (small) predefined fraction of the links, typically selected as the ones with smallest weights.
A training cycle typically consists in a full training procedure, with a complete learning rate schedule and consequently a large number of weight updates. In contrast, our work proposes to train sparse networks in a single round of gradient updates, thereby dramatically reducing the training resources. Therefore, as an original contribution, our work adopts in the forward path a soft-thresholded version of the weights that were updated based on the back-propagated gradients, and updates the parameter of the soft-threshold operator to progressively increase the ratio of pruned (inactive??) weights along the training iterations. In that way, the pruning of a weight is never definitive, and weights that are pruned at some stage of the training process get the opportunity to become non-zero again when gradient feedback makes them relevant. Moreover, gradients can be back-propagated according to the (non-zero) weights maintained along the backward path, ensuring a dense propagation of feedback, without being hampered by the definitive pruning of some links. Those specificities appear to largely benefit the performance of the trained model, compared to previous works. In particular, our proposed method is on par in terms of prediction accuracy with state-of-the-art sparse model training approaches, while significantly reducing (up to one order of magnitude) the amount of training iterations.

2 STATE OF THE ART

In this paper we concentrate our efforts on *unstructured* pruning. While pruning in a structured manner leads to an easier acceleration of CNN within existing libraries, the performance obtained at the same level of sparsity is often inferior (Li et al. (2017); He et al. (2019); Wang et al. (2020)). The work of Liu et al. (2019) also showed that the chosen structure was more important than the actual way the pruned filters were chosen, indicating that l1-based weight selection might not be the best way of choosing which filters to prune. Moreover, recent works (Kalchbrenner et al. (2018); Park et al. (2017); Gale et al. (2019)) managed to significantly speed up networks with unstructured sparsity on off-the-shelve hardware.

Naive pruning approaches remove links from the network in *one shot* at the end of an entire training cycle, based on their respective weight *magnitude* (LeCun et al. (1990); Han et al. (2015)). This however comes at the cost of a loss in prediction accuracy. In order to reduce this loss, the unpruned weights generally undergo a *finetuning* step (i.e. a shorter training cycle, with small learning rate), while the pruned ones *stay* at zero.
Over the years, many variants have been proposed to improve this baseline approach. They can be differentiated based on the way they address the three following issues:

- *Pruning criterion*: How are the pruned weights selected ?
- *Pruning and training interaction*: When is the pruning implemented along the training iterations ?
- *Pruning persistence*: Do pruned weights get the opportunity to come back to non-zero values or not ?

2.1 PRUNING CRITERION

Obviously, weights with smaller magnitude should be pruned first. However, the magnitude criterion is generally affected by two other considerations.

On the one hand, it has been widely documented (Mocanu et al. (2018); Evci et al. (2019); Lee et al. (2021)) that the network prediction accuracy is more impacted when pruning occurs close to the input. More generally, the allocation of computational resources, and thus of non-zero weights, across layers remains an open question.
Two heuristics have been recently proposed to address this issue in a satisfying manner:

1. Erdos-Rényi-Kernel (ERK) Evci et al. (2019) proposes to scale the global pruning ratio with a layer-wise pruning ratio, defined according to the number of neurons (n), the width (w) and the height (h) of the layer convolutional kernel. Formally, letting l denote the layer index, the scaling factor is defined to be $1 - \frac{n^{l-1}+n^l+w^l+h^l}{n^{l-1}\cdot n^l \cdot w^l \cdot h^l}$, which results in a more aggressive pruning of layers with more parameters.
2. Layer-Aware-Pruning (LAMP) Lee et al. (2021) selects the weights to prune globally, but relies on a score that is assigned to each weight rather than on its magnitude. This score is computed on the sorted and flattened array of weights ($\mathbf{W^1}_{\text{sorted}}$) of each layer ($l$). Mathematically, $\text{score}(i, \mathbf{W^1}_{\text{sorted}}) = \frac{(W^l_{\text{sorted},i})^2}{\sum_{j \geq i}(W^l_{\text{sorted},j})^2}$

The ERK criterion has more rigid constraints and performs slightly worse in traditional one-cycle pruning than LAMP (Lee et al. (2021)). Hence, we use the latter in Section 4.3.3 as a comparison point to our method.

2.2 PRUNING AND TRAINING INTERACTION

Pruning should only be considered after some training iterations, to make sure only the links that do not significantly impact the prediction are removed from the network. Recent literature has however revealed that the best performance are not obtained simply by pruning a fully trained network. The recent discoveries in that respect can be summarized as follows :

Subnetworks. It has been discovered by Frankle & Carbin (2019) and Frankle et al. (2020) that so-called *subnetworks*, corresponding to weights that are deemed essential for the final prediction, are formed quite early during training. This phenomenon arises because the weights change more significantly during the first training iterations than in subsequent ones (Leclerc & Madry (2020); Frankle et al. (2020)). This is simply brought about by the gradient magnitude generally starkly decreasing after a few initial iterations, denoted T_{crit} in the following, to progressively stabilize. Interestingly, recent works related to pruning have shown that it is advantageous to derive sparse networks from the weights available in the early stages of training, once the network has achieved this relative stability after this small number T_{crit} of training iterations (You et al. (2019)). This is done recursively as described below.

Rewinding. After an initial training cycle, the weights with the lowest magnitudes are pruned as before, but the remaining ones are *rewound* to their values at T_{crit} (Frankle et al. (2020)). A novel training cycle is then launched from T_{crit} onwards: meaning that the learning rate (and optimizer parameters) are reset to their values at T_{crit}, and that a full cycle of training iterations, with complete learning rate schedule, is run. The difference compared to the initial training cycle lies in the fact that the pruned weights that will stay at zero for the remainder of the restarted training cycle. Surprisingly, the sparse model obtained with this two-cycles training strategy appears to perform better than the reference baseline, obtained by finetuning the pruned weights.
In addition, the work in (Frankle et al. (2020)) has shown that applying the pruning earlier during the first training cycle leads to worse performance, thereby justifying the cost of the preliminary training cycle.
Later work by Renda et al. (2020) shows that the critical component of the rewinding step is the rewinding of the *learning rate*. In our method, the weights are constantly updated instead of being pruned or rewound at the end of a training cycle. However, as will be shown in Section 4, a rewind of the learning rate and weights can further boost the performance of our method, although coming at the cost of an additional training cycle.

Recursive Pruning with rewinding. As illustrated in our experiments (in Section 4), the prediction accuracy of the model obtained with the two-cycles training strategy sharply decreases when the pruning ratio increases. Implementing the pruning procedure *recursively* has been shown to effectively circumvent this issue: instead of pruning the whole ratio of weights after the initial training cycle, only a fraction of this ratio is actually pruned. The surviving weights are rewound, the training cycle resumes from T_{crit} and, when it ends, an *additional* fraction of the weights are pruned. The remaining weights are then rewound, training restarts and so on. This procedure results in the state-of-the-art performance, but rapidly gets expensive in terms of training time. If we take a 20% step-size (in order to compare with Frankle & Carbin (2019)), 11 training cycles are necessary to reach $\sim 90\%$ sparsity.

2.3 Pruning persistence

Above, once a weight has been pruned, it *stays* pruned all along the training cycle. Some works however have considered the possibility to re-activate a weight that has been set to zero. Works like Dettmers & Zettlemoyer (2019) and Evci et al. (2019) rely on computations on the gradient magnitudes to decide to re-activate weights. In contrast, Kusupati et al. (2020) learns a progressive layer-wise threshold via back-propagation, but only computes the *backpropagated* gradients on the *unpruned* weights. In contrast, our work allows for all gradients to flow through, updating even weights that didn't take part in the forward computation.

3 Our pruning method

To train a sparse network in a single training cycle without penalizing its prediction accuracy, our work proposes to adopt a non-persistent soft-thresholded pruning strategy and to progressively increase the sparsity ratio of the network along the iterations of a training cycle. The remainder of this section presents and motivates the 3 main components of our method. Our ablation study in Section 4.3 will confirm our arguments experimentally.

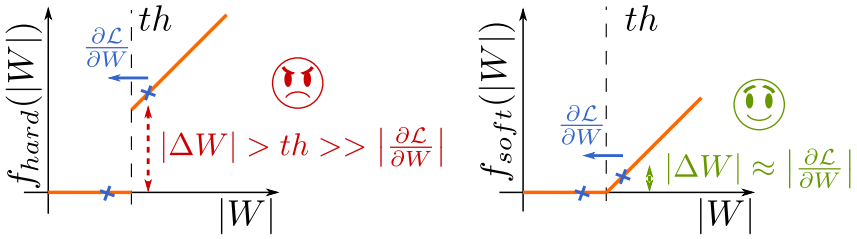

Figure 1: Example showing that hard thresholding might lead to a change in weight in the forward path that is bigger than the gradient magnitude. In contrast, soft-thresholding preserves a smooth evolution of the forward weight compared to the one maintained in the backward path.

Non-persistent pruning In contrast to most previous works, our approach proposes to maintain two versions of the weights. A dense version of the weights is maintained and updated in the backward path, while their thresholded version is considered in the forward path and to backpropagate the gradients, so as to mimick the impact of pruning. This differentiation of weights in the forward and backward paths is inspired by the straight-through-estimator used to train quantized neural networks (Hubara et al. (2016); Mellempudi et al. (2017)). Formally, we define the sparse weights W_{sparse} adopted in the forward path as a thresholded version of the dense weights W_{dense} updated in the backward path: $W_{parse} = \text{apply\_th}(W_{dense}, th)$. A consequence of this definition of W_{sparse} is that a forward weight that has been zeroed at some point during training might become non-zero again if its corresponding backward weight magnitude sufficiently increases in subsequent training iterations. Hence, the pruning emulated in the forward path is non-persistent.

Soft Thresholding Most of the persistent pruning literature makes use of hard thresholding (Han et al. (2015); Frankle & Carbin (2019)). However, in presence of non-persistent pruning, the weights that are maintained in the backward path, are likely to cross the pruning threshold, and be subject to the hard thresholding function discontinuity. This might lead to abrupt changes in the forward path, which might be inconsistent with the update implemented in the backward path. Figure 1 depicts a case where the pruning threshold is large compared to the gradient magnitude. In this case, hard thresholding induces severe discontinuities in the forward weight update, while soft thresholding preserves a smooth and consistent evolution of this forward weight. Our experiments in Section 4.3.2 confirm that soft thresholding results in a more stable training.

Increasing the pruning ratio To mitigate the instabilities induced by pruning, and leave the network the time to adapt to the zeroing of some weights in the forward path, we progressively increase the sparsity ratio during training. This allows the remaining weights to more easily adapt to the removal of a majority of its connections and avoids the neural network from getting stuck in a local minimum early on. A slow increase of the sparsity ratio during training was already applied successfully by Kusupati et al. (2020) and Wang et al. (2020). The former used it to approximately regulate the sparsity of each layer independently, while the latter applied it to the regularization of small weights, forcing them closer to zero over time. We use this linearly increasing ratio to force an ever-increasing *exact* amount of sparsity into our network. This ratio will increase linearly during the first half of the training, which also corresponds to the highest learning rate plateau for the networks trained on Cifar-10. A pseudo-code for our method is provided in Appendix A.1.

4 Experiments

4.1 Experimental setup and terminology

Our experiments train ResNet-20 (He et al. (2015)) and VGG-13 (Simonyan & Zisserman (2014)) (as simplified by Liu et al. (2019)) on the Cifar-10 Krizhevsky (2009) dataset. Every training cycle uses the same hyperparameters as defined in Appendix A.2 regardless of the method. This follows the choices made in Frankle et al. (2020) and, in practice, no significant difference has been observed when tuning the parameters individually for each method. For each pruning method, each network is trained three times, with distinct initialization seeds. The same triplet of seeds is used

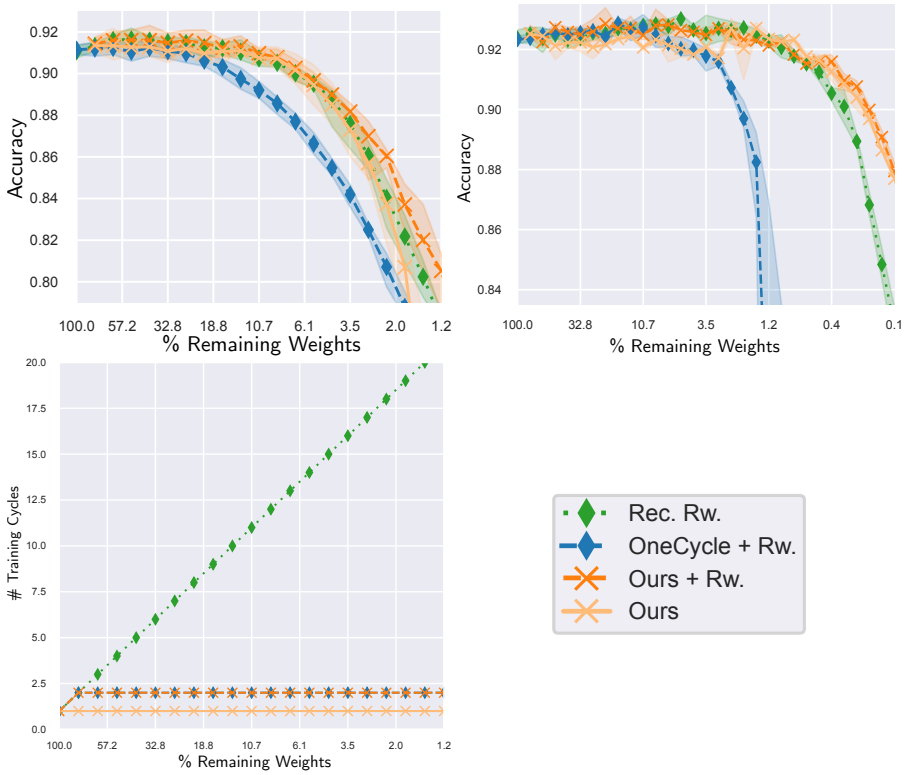

Figure 2: (Best viewed in colour) Top: Mean accuracy and variance, from 3 different initialization seeds, for a ResNet-20 (left) and VGG-11 (right) on Cifar10 as a function of the sparsity ratio for our method with (dark orange) or without rewind (orange) compared to recursive rewinding (green) and a 1-cycle rewind (blue). Bottom: Number of training cycles necessary for the 4 different methods in function of the pruning ratio

Ours		Kusupati et al. (2020)	
Accuracy	Sparsity	Sparsity	Accuracy
74.52	90.00	**90.23**	74.31
72.32	**95.00**	94.80	70.97
66.18	**98.00**	97.78	62.84
54.92	**99.00**	98.98	51.82

Table 1: Comparison of our method with Kusupati et al. (2020) applied on a ResNet-50 on ImageNet (best performance are in bold)

for all pruning methods, and all figures display the mean and variance of the accuracy achieved by the models obtained from those seeds as a function of the sparsity ratio.

To denote the methods compared in this section, we introduce the following terms:

1. OneCycle : the network is pruned after one training cycle
2. Rw (rewind): an additional training is applied to the sub-network pruned at the end of the initial (or previous, in case of recursive pruning) training cycle. The weights, learning rate and optimizer of this subnetwork are rewound to T_{crit}.
3. Rec. Rw (recursive rewind) : the pruning/rewind/training procedure is applied recursively, by pruning 20% of the remaining weights at each recursive step.
4. SoftThresh (Ours): As explained in Section 3, an increasing fraction of weights is set to zero in the forward path, by soft-thresholding the weights maintained in the backward path. The soft-threshold operator parameter is defined to progressively increase the sparsity ratio of the weights along the full training cycle.

4.2 Comparison with baselines

Pruning baseline Recursive rewind corresponds to the SotA in terms of pruned network accuracy, while one-cycle with rewind corresponds to a less accurate but also less complex (in terms of number of training cycles) baseline. Figure 2 plots our method against those 2 baselines. Our method matches recursive rewind all the while needing exponentially less training cycles.

Prior work To the best of our knowledge, Kusupati et al. (2020) detains the SotA one-cycle pruning results with a ResNet-50 on ImageNet on all thresholds. Table 1 compares our method, which was trained with the same meta-parameters. Our approach exceeds the performance of Kusupati et al. (2020) at approximately equivalent pruning budgets.

4.3 Ablation Study

This section considers variants of our method to answer the following 3 questions:

1. Are dense weights necessary for gradient computation?
2. Is soft-thresholding better than hard-thresholding in practice?
3. Does the pruning criterion, i.e. the selection of the soft-thresholding operator parameter for each layer, affect performance?

4.3.1 Dense or sparse?

As stated earlier in Section 3, we update the *dense* weights during back-propagation. To emphasize the importance of this dense update strategy, our results are compared to the *sparse* update of weights during the backward pass. In that case, only the weights whose magnitude is larger than the thresholding operator parameter are updated. In Figure 3, these 2 scenarios (respectively referred to

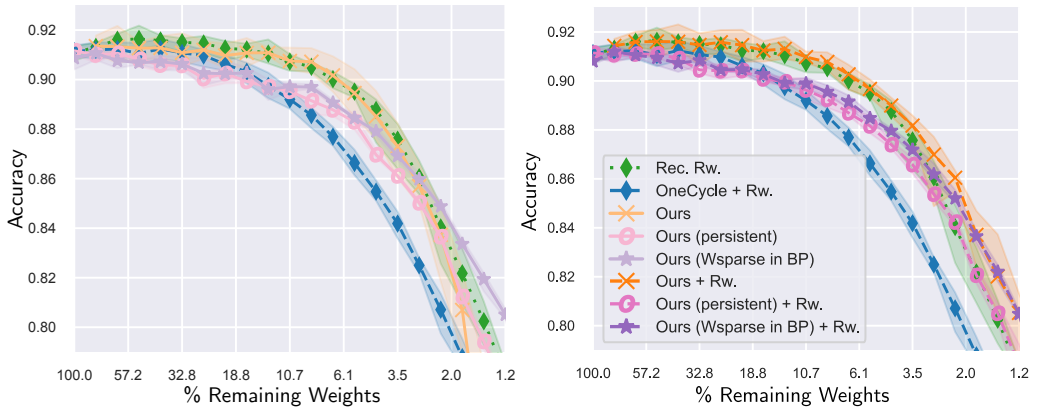

Figure 3: (Best viewed in colour) Mean accuracy and variance, from 3 different initialization seeds, for a ResNet-20 on Cifar10 as a function of the sparsity ratio for different gradient densities. Results are shown without (left) and with (right) and additional rewind cycle. Recursive rewind and one-cycle act as a frame of reference.

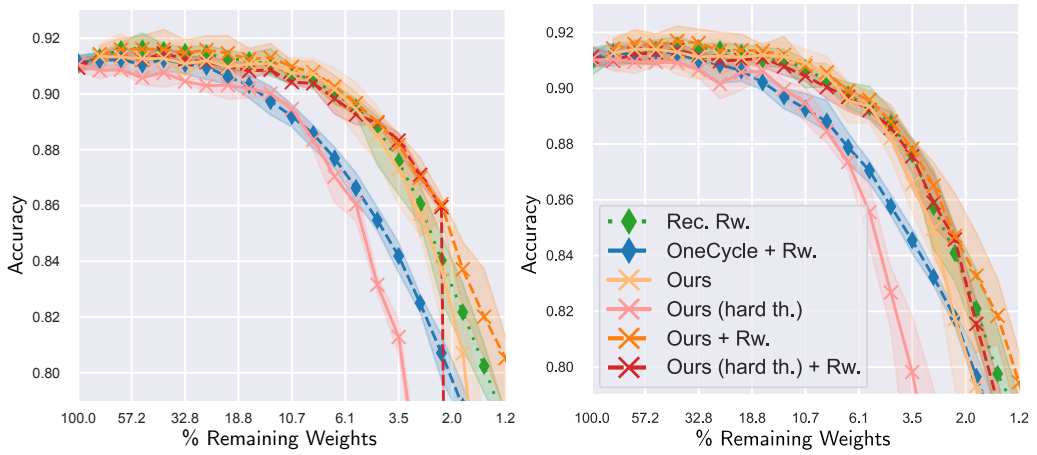

Figure 4: (Best viewed in colour) Mean accuracy and variance, from 3 different initialization seeds, for a ResNet-20 on Cifar10 as a function of the sparsity ratio for a global l1 pruning criterion (left) and LAMP (right).

as *persistent* and *Wsparse in BP*) were inspected, and a clear trend emerges : updating only the active weights during back-propagation leads to a lower accuracy compared to updating all the weights (dense).

4.3.2 Hard or Soft?

In Section 3, an explanation was given as to why a soft-thresholded approach may lead to better results than a hard-thresholded one due to the removal of a discrepancy between forward and backward propagation. Figure 4 clearly confirms our intuition. We note that until $\approx 98\%$ sparsity both strategies are relevant to threshold the weights: The hard- and soft-threshold performance *after* weight rewinding and re-training is nearly identical. However, the hard-threshold training barely manages to keep up with our baseline of one-cycle+rewind. This is caused by a highly fluctuating loss function during training, implying a great deal of instability. Moreover, this instability reaches a critical point when only $\approx 2\%$ of the weights remain, after which the network becomes unable to learn anything due to a whole layer being pruned.

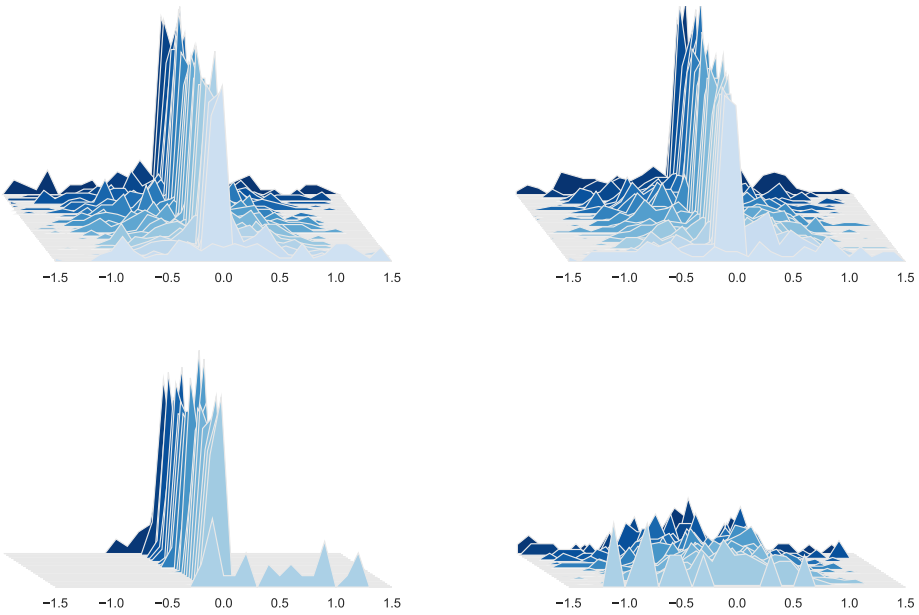

Figure 5: (Best viewed in colour) Weight histograms for ResNet-20 on Cifar-10 at 98.8% sparsity, trained with different methods. (Top) Pruning methods were used: OneCycle+rewind (left) and recursive (right). (Bottom) Our soft-thresholding method (left) and the follow-up result after rewinding and retraining (right). For clarity's sake, weights equal to zero have been removed of the histogram.

4.3.3 L1 OR LAMP?

Criteria like LAMP (Lee et al. (2021)) try to mitigate instabilities by normalizing individual weights with respect to the other ones inside their layer. To the right in Figure 4 LAMP is shown to significantly improve the results of hard thresholding at high pruning ratio, getting rid of the noted instability.

5 WEIGHT INSIGHTS

The previous sections have numerically shown the importance of keeping the dense version of the gradient when updating the weights through gradient descent. By taking a look at the underlying weight distributions and zeroed weights, two additional observations stand out:

1. The weight distributions that result from our soft-thresholding method, are different from those generated by recursive pruning
2. During training our method encourages a high number of weights to flip between a zero (inactive) and non-zero (active) state. This high number of crossings leads to networks that generalize better.

5.1 WEIGHT DISTRIBUTIONS

The weight distributions for all the layers are depicted in Figure 5 at a sparsity ratio of 98.8% to more clearly see the emerging phenomena. The weight distributions generated by *pruning* methods (be it OneCycle+rewind or Recursive-rewind) differ heavily from those obtained with our soft-thresholded approach. Our initial training cycle generates a very peaky normal distribution. Rewinding the weights and only retraining the unpruned ones, puts us closer to distributions resembling the sum of 2 overlapping gaussians mirrored around zero. This is in opposition to pruning which generates a mostly normal distribution with some weights that are pushed to higher absolute values. This indicates that our method selects weights to set to zero in a fundamentally different way than traditional pruning.

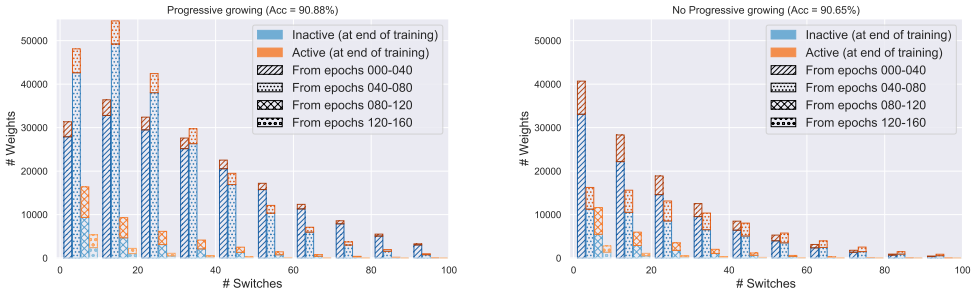

Figure 6: (Best viewed in colour) Number of threshold crossings for a ResNet-20 trained on Cifar-10 at 86.6% sparsity grouped into sets of 40 epochs. (left) the sparsity ratio grows linearly from T_{crit} to epoch 80 (right) full sparsity ratio is used from T_{crit} onwards. Weights that undergo one or no threshold crossings are removed for readability purposes.

5.2 Threshold crossings

In Section 3, a plausible explanation was given for the use of growing sparsity. However, we can observe why this training regime is so beneficial: growing the sparsity ratio leads to more weights jumping from an inactive to an active state and vice-versa. This phenomenon will be referred to as threshold crossings. Figure 6 shows the number of times the weights will cross during different training phases that are each 40 epochs long. The weights that will end up staying inactive once the training completed, are separated to more clearly show the trend: a lot of crossings occur during the progressive growing of the sparsity ratio before stabilizing after 80 epochs.

This same experience was repeated using the whole sparsity ratio from T_{crit} onwards in order to eliminate the high learning rate factor. Not allowing this ratio to progressively grow over time, leads first to a noticeably lower final accuracy and second we observe fewer threshold crossings during the early phases of the training process. This implies that the mask is chosen at T_{crit} and doesn't change much from there on out, moving our method closer to real pruning than is desirable.
This observation points to the weights needing some time to adapt to the disappearance of their neighbours. This also consolidates our method's starting point that applying fewer constraints on the weight updates, thanks to dense gradients, leads to the spontaneous emergence of weight-adjustments.

6 Conclusion

We have proposed to apply soft-thresholding to the weights maintained in the network backward path, so as to define sparse weights in the forward path. The soft-thresholding operator parameter is changed along training to progressively increase the ratio of weights that are set to zero by the thresholding, thereby reaching high sparsity ratio in a single training cycle. This is in contrast to traditional persistent pruning methods that need many training cycles to obtain models whose prediction accuracy reaches what our method obtains in one shot. Our analysis has shown the importance of using soft-thresholding, and not hard-thresholding, to avoid sharp discrepancies in the evolution of the weights maintained in the forward and backward paths. The lesser constraints of these dense gradients on the weights allow for threshold crossings which validate the importance of our method. Overall, our method results in a new SotA for one-training-cycle accuracy on ImageNet with a ResNet-50 of 66.18% at 98% sparsity.

Reproducibility Statement

The source code for this paper has been added to the supplementary material of this submission and the code will be open-sourced after publication. Every training configuration has been safeguarded in order to minimize any possible misinterpretation of the manuscript.
The network architectures used are standardized (ResNet, VGG) and so are the datasets (Cifar-10, ImageNet). The training mete-parameters were taken from previous in order to enable a better comparison accross the literature.
Whenever it was computationally sound, experiments were run with 3 different seeds (shared across all experiments) in order to guarantee more consistent results.

REFERENCES

Tim Dettmers and Luke Zettlemoyer. Sparse Networks from Scratch: Faster Training without Losing Performance. (1):1–14, 2019. URL http://arxiv.org/abs/1907.04840.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the Lottery: Making All Tickets Winners. pp. 1–21, 2019. URL http://arxiv.org/abs/1911.11134.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–42, 2019.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we Missing the Mark? *arXiv*, pp. 1–29, 2020. ISSN 23318422.

Trevor Gale, Erich Elsen, and Sara Hooker. The State of Sparsity in Deep Neural Networks. 2019. URL http://arxiv.org/abs/1902.09574.

Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. pp. 1–14, 2015. ISSN 01406736. doi: abs/1510.00149/1510.00149. URL http://arxiv.org/abs/1510.00149.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. 2015. ISSN 1664-1078. doi: 10.3389/fpsyg.2013.00124. URL http://arxiv.org/abs/1512.03385.

Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:4335–4344, 2019. ISSN 10636919. doi: 10.1109/CVPR.2019.00447.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. 2016. ISSN 1664-1078. doi: 10.1109/CVPR.2016.90. URL http://arxiv.org/abs/1609.07061.

Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimber, Aäron Van Den Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient neural audio synthesis. *35th International Conference on Machine Learning, ICML 2018*, 6: 3775–3784, 2018.

Alex Krizhevsky. (CIFAR10) Learning Multiple Layers of Features from Tiny Images. ... *Science Department, University of Toronto, Tech.* ..., pp. 1–60, 2009. ISSN 1098-6596. doi: 10.1.1.222.9220. URL http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Learning+Multiple+Layers+of+Features+from+Tiny+Images#0.

Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. *arXiv*, 2020. ISSN 23318422.

Guillaume Leclerc and Aleksander Madry. The Two Regimes of Deep Network Training. 2020. URL http://arxiv.org/abs/2002.10376.

Yann LeCun, John S Denker, and Sara A. Solla. Optimal Brain Damage (Pruning). *Advances in neural information processing systems*, pp. 598–605, 1990.

Jacho Lee, Sejun Park, Sangwoo Mo, Ahn Sungsoo, and Jinwoo Shin. Layer-Adaptive Sparsity for the Magnitude-Based Pruning. *International Conference on Learning Representations*, (2019): 1–19, 2021.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H.S. Torr. SnIP: Single-shot network pruning based on connection sensitivity. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

Hao Li, Hanan Samet, Asim Kadav, Igor Durdanovic, and Hans Peter Graf. Pruning filters for efficient convnets. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, (2016):1–13, 2017.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–21, 2019.

Naveen Mellempudi, Abhisek Kundu, Dheevatsa Mudigere, Dipankar Das, Bharat Kaul, and Pradeep Dubey. Ternary Neural Networks with Fine-Grained Quantization. 2017. URL http://arxiv.org/abs/1705.01462.

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*, 9(1), 2018. ISSN 20411723. doi: 10.1038/s41467-018-04316-3.

Jongsoo Park, Hai Li, Sheng Li, Wei Wen, Yiran Chen, Ping Tak Peter Tang, and Pradeep Dubey. Faster cnns with direct sparse convolutions and guided pruning. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2017:1–12, 2017.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing Rewinding and Fine-tuning in Neural Network Pruning. (2019):1–31, 2020. URL http://arxiv.org/abs/2003.02389.

K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, 2014.

Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural Pruning via Growing Regularization. pp. 1–15, 2020. URL http://arxiv.org/abs/2012.09243.

Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Richard G. Baraniuk, Zhangyang Wang, and Yingyan Lin. Drawing early-bird tickets: Towards more efficient training of deep networks. (2019):1–13, 2019. URL http://arxiv.org/abs/1909.11957.

A APPENDIX

A.1 PSEUDO-CODE

Listing 1: Pytorch-like pseudo-code for Recursive pruning with rewind

```
act_pruning = 0
W = Winit
Wmask = ones_like(W)
train_cycles = 0
while True:
    # Complete Training cycle
    for epoch in range(n_epochs):
            for inputs, true_outputs in dataset:
            outputs = NN(W * Wmask)(inputs)
            loss = loss_fx(true_outputs, outputs)
            compute_gradients(loss, W[Wmask==1])
            W = optimizer.update(W[Wmask==1], learning_rate)
        learning_rate.update(epoch)

    act_pruning = 1 - (1-pruning_step)^train_cycles
    if act_pruning > final_pruning:
        break

    # Prune more weights
    Wmask = Wmask - prune(W[Wmask==1], pruning_step)
```

Listing 2: Pytorch-like pseudo-code for our soft-thresholded sparse training with increasing sparsity

```
W = Winit
start_epoch = Tcrit
end_epoch = n_epochs // 2
pruning_delta = final_pruning / (len(dataset)*(end_epoch-start_epoch))
act_pruning = 0
# Complete Training cycle
for epoch in range(n_epochs):
    for inputs, true_outputs in dataset:
        th = compute_prune_th(W, act_pruning)

        # Do not compute gradients on pruning offset
        with no_gradients():
            Wdelta = soft_threshold(W, th) - W

        Wpruned = W + Wdelta
        outputs = NN(Wpruned)(inputs)
        loss = loss_fx(true_outputs, outputs)
        compute_gradients(loss, W)
        W = optimizer.update(W, learning_rate)

        if epoch >= start_epoch and epoch < end_epoch:
            act_pruning += pruning_delta
    learning_rate.update(epoch)
```

Network	ResNet-20	VGG-13
Optimizer	SGD	
Lr	0.1	0.2
Momentum	0.9	
Weight-decay	1e-4	
Batch-size	128	
#epochs	160	
Lr-stepdown	[80,120]; lr*0.1	

Table 2: Hyperparameters used for training

A.2 HYPER-PARAMETER TUNING

The hyper parameters used for ResNet-20 and VGG-13 trained on Cifar-10 are displayed in Table 2. All run configurations are also available in the code submitted with the supplementary material.