Sparse Training via Boosting Pruning Plasticity with Neuroregeneration

Shiwei Liu[1]*, Tianlong Chen[2], Xiaohan Chen[2], Zahra Atashgahi[3], Lu Yin[1], Huanyu Kou[4], Li Shen[5], Mykola Pechenizkiy[1,6], Zhangyang Wang[2], Decebal Constantin Mocanu[1,3]
[1]Eindhoven University of Technology, [2]University of Texas at Austin
[3]University of Twente,[4]University of Leeds,[5]JD Explore Academy, [6]University of Jyväskylä
{s.liu3,l.yin,m.pechenizkiy}@tue.nl, {tianlong.chen,xiaohan.chen,atlaswang}@utexas.edu
{z.atashgahi,d.c.mocanu}@utwente.nl, {khydouble1,mathshenli}@gmail.com

Abstract

Works on lottery ticket hypothesis (LTH) and single-shot network pruning (SNIP) have raised a lot of attention currently on post-training pruning (iterative magnitude pruning), and before-training pruning (pruning at initialization). The former method suffers from an extremely large computation cost and the latter usually struggles with insufficient performance. In comparison, during-training pruning, a class of pruning methods that simultaneously enjoys the training/inference efficiency and the comparable performance, temporarily, has been less explored. To better understand during-training pruning, we quantitatively study the effect of pruning throughout training from the perspective of **pruning plasticity** (the ability of the pruned networks to recover the original performance). Pruning plasticity can help explain several other empirical observations about neural network pruning in literature. We further find that pruning plasticity can be substantially improved by injecting a brain-inspired mechanism called **neuroregeneration**, i.e., to regenerate the same number of connections as pruned. We design a novel gradual magnitude pruning (GMP) method, named gradual pruning with zero-cost neuroregeneration (**GraNet**), that advances state of the art. Perhaps most impressively, its sparse-to-sparse version for the first time boosts the sparse-to-sparse training performance over various dense-to-sparse methods with ResNet-50 on ImageNet without extending the training time. We release all codes in https://github.com/Shiweiliuiiiiiii/GraNet.

1 Introduction

Neural network pruning is the most common technique to reduce the parameter count, storage requirements, and computational costs of modern neural network architectures. Recently, post-training pruning [49, 29, 18, 47, 10, 54, 74, 5, 57, 75] and before-training pruning [31, 30, 67, 63, 6, 11] have been two fast-rising fields, boosted by lottery tickets hypothesis (LTH) [10] and single-shot network pruning (SNIP) [31]. The process of post-training pruning typically involves fully pre-training a dense network as well as many cycles of retraining (either fine-tuning [18, 17, 39] or rewinding [12, 54]). As the training costs of the state-of-the-art models, e.g., GPT-3 [4] and FixEfficientNet-L2 [64] have exploded, this process can lead to a large amount of overhead cost.

Recently emerged methods for pruning at initialization significantly reduce the training cost by identifying a trainable sub-network before the main training process. While promising, the existing methods fail to match the performance achieved by the magnitude pruning after training [11].

*Partial of this work have been done when Shiwei Liu worked as an intern at JD Explore Academy.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

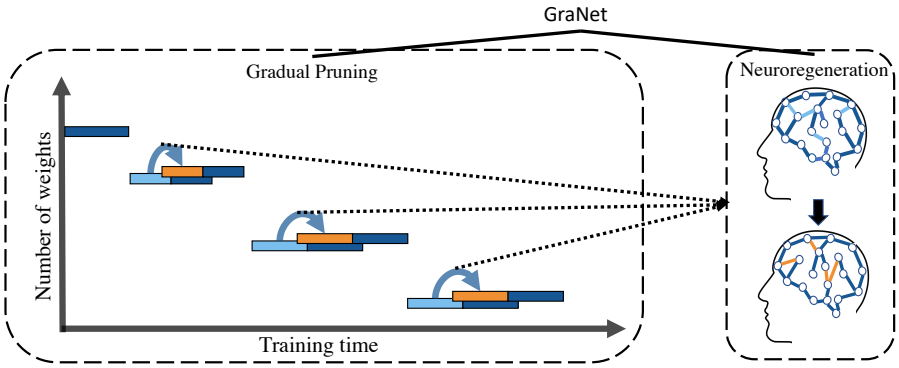

Figure 1: **Schematic view of GraNet. Left:** Gradual pruning starts with a sparse subnetwork and gradually prune the subnetwork to the target sparsity during training. **Right:** We perform zero-cost neuroregeneration after each gradual pruning step. Light blue blocks/lines refer to the "damaged" connections and orange blocks/lines refer to the regenerated new connections.

Compared with the above-mentioned two classes of pruning, during-training pruning is a class of methods that reap the acceleration benefits of sparsity early on the training and meanwhile achieve promising performance by consulting the information obtained during training. There are some works [77, 13, 33] attempting to gradually prune the network to the desired sparsity during training, while they mainly focus on the performance improvement. Up to now, the understanding of during-training pruning has been less explored due to its more complicated dynamical process, and the performance gap still exists between pruning during training and full dense training.

To better understand the effect of pruning during the optimization process (not at inference), we study the ability of the pruned models to recover the original performance after a short continued training with the *current* learning rate, which we call *pruning plasticity* (see Section 3.1 for a more formal definition). Inspired by the *neuroregeneration* mechanism in the nervous system where new neurons and connections are synthesized to recover the damage in the nervous system [26, 41, 73], we examine if allowing the pruned network to regenerate new connections can improve pruning plasticity, and hence contribute to pruning during training. We consequently propose a parameter-efficient method to regenerate new connections during the gradual pruning process. Different from the existing works for pruning understanding which mainly focus on dense-to-sparse training [42] (training a dense model and prune it to the target sparsity), we also consider sparse-to-sparse training (training a sparse model yet adaptively re-creating the sparsity pattern) which recently has received an upsurge of interest in machine learning [44, 3, 9, 48, 8, 37, 36].

In short, we have the following main findings during the course of the study:

#1. Both pruning rate and learning rate matter for pruning plasticity. When pruned with low pruning rates (e.g., 0.2), both dense-to-sparse training and sparse-to-sparse training can easily recover from pruning. On the contrary, if too many parameters are removed at one time, almost all models suffer from accuracy drops. This finding makes a connection to the success of the iterative magnitude pruning [10, 54, 5, 6, 65], where usually a pruning process with a small pruning rate (e.g., 0.2) needs to be iteratively repeated for good performance.

Pruning plasticity also gradually decreases as the learning rate drops. When pruning happens during the training phase with large learning rates, models can easily recover from pruning (up to a certain level). However, pruning plasticity drops significantly after the second learning rate decay, leading to a situation where the pruned networks can not recover with continued training. This finding helps to explain several observations (1) for gradual magnitude pruning (GMP), it is always optimal to end pruning before the second learning rate drop [77, 13]; (2) dynamic sparse training (DST) benefits from a monotonically decreasing pruning rate with cosine or linear update schedule [8, 9]; (3) rewinding techniques [12, 54] outperform fine-tuning as rewinding retrains subnetworks with the original learning rate schedule whereas fine-tuning often retrains with the smallest learning rate.

#2. Neuroregeneration improves pruning plasticity. Neuroregeneration [41, 73] refers to the regrowth or repair of nervous tissues, cells, or cell products. Conceptually, it involves synthesizing new neurons, glia, axons, myelin, or synapses, providing extra resources in the long term to replace

those damaged by the injury, and achieving a lasting functional recovery. Such mechanism is closely related to the brain plasticity [51], and we borrow this concept to developing a computational regime.

We show that, while regenerating the same number of connections as pruned, the pruning plasticity is observed to improve remarkably, indicating a more neuroplastic model being developed. However, it increases memory and computational overheads and seems to contradict the benefits of pruning-during-training. This however raises the question: *can we achieve efficient neuroregeneration during training with no extra costs*? We provide an affirmative answer to this question.

#3. Pruning plasticity with neuroregeneration can be leveraged to substantially boost sparse training performance. The above-mentioned findings of pruning plasticity can generalize to the final performance level under a full continued training to the end. Imitating the neuroregeneration behavior [41, 73], we propose a new sparse training method – gradual pruning with zero-cost neuroregeneration (GraNet), which is capable of performing regeneration without increasing the parameter count.

In experiments, GraNet establishes the new state-of-the-art performance bar for dense-to-sparse training and sparse-to-sparse training, respectively. Particularly, the latter for the first time boosts the sparse-to-sparse training performance over various dense-to-sparse methods by a large margin without extending the training time, with ResNet-50 on ImageNet. Besides the consistent performance improvement, we find the subnetworks that GraNet learns are more accurate than the ones learned by the existing gradual pruning method, providing explanations for the success of GraNet.

2 Related Work

Post-Training Pruning. Methods that yield a sparse neural network from a pre-trained network by pruning the unimportant weights or neurons, to the best of our knowledge, were proposed in [24] and [50]. After that, various pruning methods have emerged to provide increasingly efficient methods to identify sparse neural networks for inference. The pruning criterion includes weight magnitude [18, 10], gradient [61] Hessian [29, 19, 59], Taylor expansion [47, 46], etc. Low-rank decomposition [7, 23, 17, 71] are also used to induce structured sparsity in terms of channels or filters. Most of the above-mentioned pruning methods require many pruning and re-training cycles to achieve the desired performance.

During-Training Pruning. Instead of inheriting weights from a pre-trained model, some works attempt to discover well-performing sparse neural networks with one single training process.

Gradual Magnitude Pruning (GMP), introduced in [77] and studied further in [13], gradually sparsifies the neural network during the training process until the desired sparsity is reached. Besides, [40] and [68] are prior works that enforce the network to sparse during training via L_0 and L_1 regularization, respectively. [60, 34, 55, 70, 28] moved further by introducing trainable sparsity heuristics to learn the sparse masks and weights simultaneously. These methods are all classified as dense-to-sparse training as they start from a dense network.

Dynamic Sparse Training (DST) [44, 3, 48, 8, 9, 36, 35, 25] is another class of methods that prune models during training. The key factor of DST is that it starts from a random initialized sparse network and optimizes the sparse topology as well as the weights simultaneously during training (sparse-to-sparse training). Without an extended training time [37], sparse-to-sparse training usually falls short of dense-to-sparse training in terms of the prediction accuracy. For further details, see the survey of [43, 21].

Before-Training Pruning. Motivated by SNIP [31], many works [67, 63, 6] have emerged recently to explore the possibility of obtaining a trainable sparse neural network before the main training process. [11] demonstrates that the existing methods for pruning at initialization perform equally well when the unpruned weights are randomly shuffled, which reveals that what these methods discover is the layer-wise sparsity ratio, rather than the indispensable weight values and positions. Our analysis shows that both the mask positions and weight values are crucial for GraNet.

3 Methodology for Pruning Plasticity

The primary goal of this paper is to study the effect of pruning as well as neuroregeneration on neural networks during the standard training process. Therefore, we do not consider post-training pruning and before-training pruning. Below, we introduce in detail the definition of pruning plasticity and the experimental design that we used to study pruning plasticity.

3.1 Metrics

Let us denote $W_t \in \mathbb{R}^d$ as the weights of the network and $m_t \in \{0,1\}^d$ as the binary mask yielded from the pruning method at epoch t. Thus, the pruned network can be denoted as $W_t \odot m_t$. Let T be the total number of epochs the model should be trained. Let $\text{CONTRAIN}^k(W_t \odot m_t, a)$ refers to the function that continues to train the pruned model for k epochs with the learning rate schedule a.

Definition of Pruning plasticity. We define pruning plasticity as $t_{\text{CONTRAIN}^k(W_t \odot m_t, a_t)} - t_{\text{PRE}}$, where t_{PRE} is the test accuracy measured before pruning and $t_{\text{CONTRAIN}^k(W_t \odot m_t, a_t)}$ is the test accuracy measured after k epoch of continued training $\text{CONTRAIN}^k(W_t \odot m_t, a_t)$. Specifically, to better understand the effect of pruning on the current model status and to avoid the effect of learning rate decay, we fix the learning rate as the one when the model is pruned, i.e, a_t. This setting is also appealing to GMP [77, 13] and DST [44, 9, 48, 37] in which most of the pruned models are continually trained with the current learning rate for some time.

Final performance gap. Nevertheless, we also investigate the effect of pruning on the final performance, that is, continually training the pruned networks to the end with the remaining learning rate schedule $\text{CONTRAIN}^{T-t}(W_t \odot m_t, a_{[t+1:T]})$. In this case, we report $t_{\text{CONTRAIN}^{T-t}(W_t \odot m_t, a_{[t+1:T]})} - t_{\text{FINAL}}$, where t_{FINAL} is the final test accuracy of the unpruned models.

3.2 Architectures and Datasets

We choose two commonly used architectures to study pruning plasticity, VGG-19 [58] with batch normalization on CIFAR-10 [27], and ResNet-20 [20] on CIFAR-10.

We share the summary of the networks, data, and hyperparameters of dense-to-sparse training in Table 1. We use standard implementations and hyperparameters available online, with the exception of the small batch size for the ResNet-50 on ImageNet due to the limited hardware resources (2× Tesla V100). All accuracies are in line with the baselines reported in the references [8, 11, 67, 9, 37].

Table 1: Summary of the architectures and hyperparameters we study in this paper.

Model	Data	#Epoch	Batch Size	LR	LR Decay, Epoch	Weight Decay	Test Accuracy
ResNet-20	CIFAR-10	160	128	0.1 ($\beta = 0.9$)	10×, [80, 120]	0.0005	92.41±0.04
VGG-19	CIFAR-10	160	128	0.1 ($\beta = 0.9$)	10×, [80, 120]	0.0005	93.85±0.05
	CIFAR-100	160	128	0.1 ($\beta = 0.9$)	10×, [80, 120]	0.0005	73.43±0.08
ResNet-50	CIFAR-10	160	128	0.1 ($\beta = 0.9$)	10×, [80, 120]	0.0005	94.75±0.01
	CIFAR-100	160	128	0.1 ($\beta = 0.9$)	10×, [80, 120]	0.0005	78.23±0.18
	ImageNet	100	64	0.1 ($\beta = 0.9$)	10×, [30, 60, 90]	0.0004	76.80±0.09

3.3 How to Prune, and How to Regenerate

Structured and Unstructured Pruning. We consider unstructured and structured pruning in this paper. Structured pruning prunes weights in groups, or removes the entire neurons, convolutional filters, or channels, enabling acceleration with the off-the-shelf hardware. In particular, we choose the filter pruning method used in Li et al. [32]. Unstructured sparsity is a more promising direction not only due to its outstanding performance at extreme sparsities but the increasing support for sparse operation in the practical hardware [35, 14, 52, 76, 22]. For example, Liu et al. [35] illustrated for the first time the true potential of DST, demonstrating significant training/inference efficiency improvement over the dense training. Different from prior conventions [77, 13, 33, 2] where values of the pruned weights are kept, we set the pruned weights to zero to eliminate the historical information for all implementations in this paper.

Magnitude pruning. We prune the weights with the smallest magnitude, as it has evolved as the standard method when pruning happens during training, e.g., GMP [77, 13] and DST [44, 9, 37]. We are also aware of other pruning criteria including but not limited to Hessian [29, 19, 59], Taylor expansion [47, 46], connection sensitivity [31], Gradient Flow [67], Neural Tangent Kernel [38, 16].

One-shot pruning. To isolate the pruning effect at different training stages and to avoid the interaction between two iterations of pruning, we focus on one-shot pruning. Please note that iterative pruning can also be generalized in our setting, as our experimental design includes neural networks trained at various sparsities and each of them is further pruned with various pruning rates.

Layer-wise pruning and global pruning. We study both the layer-wise magnitude pruning and global magnitude pruning for pruning plasticity. Global magnitude pruning prunes different layers

together and leads to non-uniform sparsity distributions; layer-wise pruning operates layer by layer, resulting in uniform distributions.

Gradient-based regeneration. The simplest regeneration scheme is to randomly activate new connections [3, 44]. However, it would take a lot of time for random regeneration to discover the important connections, especially for the very extreme sparsities. Alternatively, gradients, including those for the connections with zero weights, provide good indicators for the connection importance. For this reason, we focus on gradient-based regeneration proposed in Rigged Lottery (RigL) [9], i.e., regenerating the same number of connections as pruned with the largest gradient magnitude.

3.4 Experimental Results

We study pruning plasticity during training with/without regeneration, for both dense training and sparse training. We report the results of ResNet-20 on CIFAR-10 with unstructured global pruning in the main body of the paper. The rest of the experiments are given in Appendix A. Unless otherwise stated, results are qualitatively similar across all networks. Concretely, we first pre-train networks at four sparsity levels, including 0, 0.5, 0.9, and 0.98. The sparse neural networks are trained with uniform distribution (i.e., all layers have the same sparsity). We further choose four pruning rates, e.g., 0.2, 0.5, 0.9, and 0.98, to measure the corresponding pruning plasticity of the pre-trained networks.

Pruning plasticity. We continue to train the pruned model for 30 epochs and report pruning plasticity in Figure 2. Overall, the learning rate schedule, the pruning rate, and the sparsity of the original models all have a big impact on pruning plasticity. Pruning plasticity decreases as the learning rate decays for all models with different sparsity levels. The models trained with a large learning rate 0.1 can easily recover, or exceed the original performance except for the extremely large pruning rate 0.98. However, the models obtained during the later training phases can recover only with the mild pruning rate choices, e.g., 0.2 (orange lines) and 0.5 (green lines).

We next demonstrate the effect of connection regeneration on pruning plasticity in the bottom row of Figure 2. It is clear to see that connection regeneration significantly improves pruning plasticity of all the cases, especially for the models that are over-pruned (purple lines). Still, even with connection regeneration, pruning plasticity suffers from performance degradation when pruning occurs after the learning rate drops.

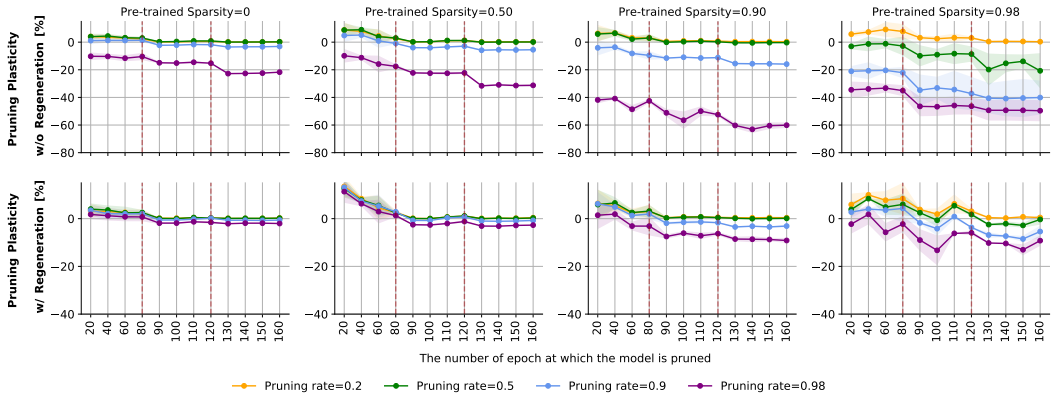

Figure 2: **Unstructured Pruning:** Pruning plasticity (see Section 3.1 for definition) under a 30-epoch continued training with and without connection regeneration for ResNet-20 on CIFAR-10. The vertical red lines refer to the points when the learning rate is decayed. "Pre-trained Sparsity" refers to the original sparsity of the pre-trained networks before pruning. The pruning method is the magnitude global pruning.

Final performance gap. Compared with the current model status, people might be more interested in the effect of pruning on the final performance. We further measure the performance gap between the original test accuracy of the unpruned models and the final test accuracy of the pruned model under a full continued training $\textsc{ConTrain}^{T-t}(W_t \odot m_t, a_{[t+1:T]})$ in Figure 3.

We observe that, in this case, large learning rates do not enjoy large performance improvement, but still, the performance gap increases as the learning rate drops. It is reasonable to conjecture that the accuracy improvement of pruning plasticity with the large learning rate, 0.1, is due to the

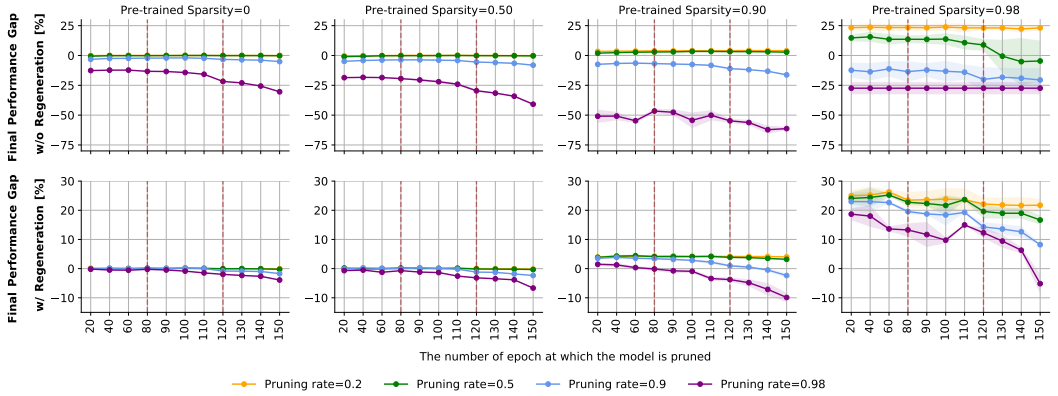

Figure 3: **Unstructured Pruning:** Final performance gap between the unpruned models and the pruned models for ResNet-20 on CIFAR-10. The vertical red lines refer to the points when the learning rate is decayed. "Pre-trained Sparsity" refers to the original sparsity of the pre-trained networks before pruning. The pruning method is the magnitude global pruning.

unconverged performance during the early phase of training. Besides, it is surprising to find that the final performance of extreme sparse networks (e.g., the third column and the fourth column) significantly benefits from mild pruning. Again, the ability of the pruned model to recover from pruning remarkably improves after regenerating the connections back.

4 Gradual Pruning with Zero-Cost Neuroregeneration

So far, we have known that regenerating the important connections to the pruned models during training substantially improves pruning plasticity as well as the final performance. However, naively regenerating extra connections increases the parameter count and conflicts with the motivation of gradual pruning.

Inspired by the mechanism of neuroregeneration in the nervous system, we propose a novel sparse training method which we call gradual pruning with zero-cost neuroregeneration (GraNet). GraNet consults the information produced throughout training and regenerates important connections during training in a parameter-efficient fashion. See Appendix B.1 for the pseudocode of GraNet. We introduce the main components of GraNet below.

4.1 Gradual Pruning

We follow the gradual pruning scheme used in [77] and gradually sparsifies the dense network to the target sparsity level over n pruning iterations. Let us define s_i is the initial sparsity, s_f is the target sparsity, t_0 is is the starting epoch of gradual pruning, t_f is the end epoch of gradual pruning, and Δt is the pruning frequency. The pruning rate of each pruning iteration is:

$$s_t = s_f + (s_i - s_f)\left(1 - \frac{t - t_0}{n\Delta t}\right)^3, \; t \in \{t_0, t_0 + \Delta t, ..., t_0 + n\Delta t\}\,. \tag{1}$$

We choose global pruning for our method as it generally achieves better performance than uniform pruning. We also report the performance of the uniform sparsity as used in [13] in Appendix C.3.

The conventional gradual pruning methods [77, 13] change the mask (not the weight values) to fulfill the pruning operation, so that the pruned connections have the possibility to be reactivated in the later training phases. Despite this, since the weights of the pruned connections are not updated, they have a small chance to receive sufficient updates to exceed the pruning threshold. This hinders the regeneration of the important connections.

4.2 Zero-Cost Neuroregeneration

The main difference between GraNet and the conventional GMP methods [77, 13] is the Zero-Cost Neuroregeneration. Imitating the neuroregeneration of the peripheral nervous system [41, 73] where new neurons and connections are synthesized to replace the damaged ones, we first detect and eliminate the "damaged" connections, and then regenerate the same number of new connections. By doing this, we can achieve connection regeneration without increasing the number of connections.

Concretely, we identify the "damaged" connections as the ones with the smallest weight magnitudes. Small magnitude indicates that either the weight's gradient is small or a large number of oscillations occur to the gradient direction. Therefore, these weights have a small contribution to the training loss and can be removed. Again, we use the gradient as the importance score for regeneration, same as the regrow method as used in RigL [9].

Why we call it "Zero-Cost Neuroregeneration"? In addition to not increasing the connection (parameter) count, the backward pass of our method is sparse most of the time even though our regeneration utilizes the dense gradient to identify the important connections. We perform neuroregeneration immediately after each gradual pruning step, meaning that the regeneration occurs only once every several thousand iterations. The extra overhead to calculate the dense gradient can be amortized compared with the whole training costs. Compared with the methods [33, 69] that require updating all the weights in the backward pass, our method is much more training efficient, as around 2/3 of the training FLOPs is owing to the backward pass [9, 72].

Let us denote r as the ratio of the number of the regenerated connections to the total number of connections; W is the network weight. We first remove r proportion of "damaged" weights with the smallest magnitude by:

$$W' = \mathrm{TopK}\left(|W|,\ 1-r\right). \tag{2}$$

Here $\mathrm{TopK}(v, k)$ returns the weight tensor retaining the top k-proportion of elements from v. Immediately after that, we regenerate r proportion of new connections based on the gradient magnitude:

$$W = W' + \mathrm{TopK}\left(|\mathbf{g}_{i \notin W'}|,\ r\right), \tag{3}$$

where $|\mathbf{g}_{i \notin W'}|$ are the gradient magnitude of the zero weights. We perform Zero-Cost Neuroregeneration layer by layer from the beginning of the training to the end.

GraNet can naturally generalize to the dense-to-sparse training scenario and the sparse-to-sparse training scenario by setting the initial sparsity level $s_i = 0$ and $s_i > 0$ in Eq. (1), respectively. For simplicity, we set $s_i = 0.5$, $t_0 = 0$, and t_f as the epoch when performing the first learning rate decay for the sparse-to-sparse training. Different from the existing sparse-to-sparse training methods, i.e., SET [44], RigL [9], and ITOP [37], in which the sparsity is fixed throughout training, GraNet starts from a denser yet still sparse model and gradually prunes the sparse model to the desired sparsity. Although starting with more parameters, the global pruning technique of gradual pruning helps GraNet quickly evolve to a better sparsity distribution than RigL with lower feedforward FLOPs and higher test accuracy. What's more, GraNet sparsifies all layers including the first convolutional layer and the last fully-connected layer.

4.3 Experimental Results

We conduct various experiments to evaluate the effectiveness of GraNet. We compare GraNet with various dense-to-sparse methods and sparse-to-sparse methods. The results of Rigged Lottery (RigL) and GMP with CIFAR-10/100 were reproduced by our implementation with PyTorch so that the only difference between GraNet and GMP is the Zero-Cost Neuroregeneration. For each model, we divide the results into three groups from top to bottom: pruning at initialization, dynamic sparse training and dense-to-sparse methods. See Appendix B for more implementation details used in the experiments. GraNet ($s_i = 0.5$) refers to the sparse-to-sparse version and the and GraNet ($s_i = 0$) refers to the dense-to-sparse version.

CIFAR-10/100. The results of CIFAR-10/100 are shared in Table 2. We can observe that performance differences among different methods on CIFAR-10 are generally small, but still, GraNet ($s_i = 0$) consistently improves the performance over GMP except for the sparsity 95%, and achieves the highest accuracy in 4 out of 6 cases. In terms of the more complex data CIFAR-100, the performance differences between the during-training pruning methods and before-training pruning methods are much larger. GraNet ($s_i = 0$) again consistently outperforms GMP with all sparsities, highlighting the benefits of Zero-Cost Neuroregeneration. It is maybe more interesting that GraNet ($s_i = 0$) even outperforms the post-training method, subdifferential inclusion for sparsity (SIS), by a large margin.

In terms of sparse-to-sparse training, our proposed GraNet ($s_i = 0.5$) has a dominant performance over other methods. Especially at the very extreme sparsity 0.98, our method outperforms RigL by 1.40% and 2.22% with VGG-19 on CIFAR-10 and CIFAR-100, respectively.

ImageNet. Due to the small data size, the experiments with CIFAR-10/100 may not be sufficient to draw a solid conclusion. We further evaluate our method with ResNet-50 on ImageNet in Table 3.

Table 2: Test accuracy of pruned VGG-19 and ResNet-50 on CIFAR-10/100. We mark the best sparse-to-sparse training results in blue and the best dense-to-sparse training results in bold. The results reported with (mean ± std) are run with three different random seeds by us. The rest are obtained from [66] and [67]. Note that the accuracy of RigL is higher than the ones reported in [66], as we choose a large update interval following the In-Time Over-Parameterization strategy [37]. s_i refers to the initial sparsity of GraNet.

Dataset	CIFAR-10			CIFAR-100		
Pruning ratio	90%	95%	98%	90%	95%	98%
VGG-19 (Dense)	93.85±0.05	-	-	73.43±0.08	-	-
SNIP [31]	93.63	93.43	92.05	72.84	71.83	58.46
GraSP [67]	93.30	93.04	92.19	71.95	71.23	68.90
SynFlow [63]	93.35	93.45	92.24	71.77	71.72	70.94
Deep-R [3]	90.81	89.59	86.77	66.83	63.46	59.58
SET [44]	92.46	91.73	89.18	72.36	69.81	65.94
RigL [9]	93.38±0.11	93.06±0.09	91.98±0.09	73.13±0.28	72.14±0.15	69.82±0.09
GraNet ($s_i = 0.5$) (ours)	**93.73±0.08**	**93.66±0.07**	**93.38±0.15**	**73.30±0.13**	**73.18±0.31**	**72.04±0.13**
STR [28]	93.73	93.27	92.21	71.93	71.14	69.89
SIS [66]	**93.99**	93.31	93.16	72.06	71.85	71.17
GMP [13]	93.59±0.10	93.58±0.07	93.52±0.03	73.10±0.12	72.30±0.15	72.07±0.37
GraNet ($s_i = 0$) (ours)	93.80±0.10	**93.72±0.11**	**93.63±0.08**	**73.74±0.30**	**73.10±0.04**	**72.35±0.26**
ResNet-50 (Dense)	94.75±0.01	-	-	78.23±0.18	-	-
SNIP [31]	92.65	90.86	87.21	73.14	69.25	58.43
GraSP [67]	92.47	91.32	88.77	73.28	70.29	62.12
SynFlow [63]	92.49	91.22	88.82	73.37	70.37	62.17
RigL [9]	94.45±0.43	93.86±0.25	93.26±0.22	76.50±0.33	76.03±0.34	75.06±0.27
GraNet ($s_i = 0.5$) (ours)	**94.64±0.27**	**94.38±0.28**	**94.01±0.23**	**77.89±0.33**	**77.16±0.52**	**77.14±0.45**
STR [28]	92.59	91.35	88.75	73.45	70.45	62.34
SIS [66]	92.81	91.69	90.11	73.81	70.62	62.75
GMP [13]	94.34±0.09	**94.52±0.08**	94.19±0.04	76.91±0.23	76.42±0.51	75.58±0.20
GraNet ($s_i = 0$) (ours)	**94.49±0.08**	94.44±0.01	**94.34±0.17**	**77.29±0.45**	**76.71±0.26**	**76.10±0.20**

Table 3: Test accuracy of pruned ResNet-50 on ImageNet dataset. The best results of DST methods are marked as blue and the best results of pruning during training methods are marked in bold. The training/test FLOPs are normalized with the FLOPs of a dense model. s_i refers to the initial sparsity of GraNet.

Method	Top-1 Accuracy	FLOPs (Train)	FLOPs (Test)	TOP-1 Accuracy	FLOPs (Train)	FLOPs (Test)
Dense	76.8±0.09	1x (3.2e18)	1x (8.2e9)	76.8±0.09	1x (3.2e18)	1x (8.2e9)
Pruning ratio		80%			90%	
Static (ERK)	72.1±0.04	0.42×	0.42×	67.7±0.12	0.24×	0.24×
Small-Dense	72.1±0.06	0.23×	0.23×	67.2±0.12	0.10×	0.10×
SNIP [31]	72.0±0.06	0.23×	0.23×	67.2±0.12	0.10×	0.10×
SET [44]	72.9±0.39	0.23×	0.23×	69.6±0.23	0.10×	0.10×
DSR[48]	73.3	0.40×	0.40×	71.6	0.30×	0.30×
RigL (ERK) [9]	75.1±0.05	0.42×	0.42×	73.0±0.04	0.25×	0.24×
SNFS (ERK) [8]	75.2±0.11	0.61×	0.42×	72.9±0.06	0.50×	0.24×
GraNet ($s_i = 0.5$) (ours)	**76.0**	0.37×	0.35×	**74.5**	0.25×	0.20×
STR [28]	**76.1**	n/a	0.17×	74.0	n/a	0.08×
DPF [33]	75.1	0.71×	0.23×	n/a	n/a	n/a
GMP [13]	75.6	0.56×	0.23×	73.9	0.51×	0.10×
GraNet ($s_i = 0$) (ours)	75.8	0.34×	0.28×	**74.2**	0.23×	0.16×
Small-Dense$_{5\times}$	73.9	1.01$_{\times}$	0.20$_{\times}$	71.3	0.60$_{\times}$	0.12$_{\times}$
RigL-ITOP$_{2\times}$ [37]	76.9	0.84×	0.42×	75.5	0.50×	0.24×
RigL$_{5\times}$ [9]	**77.1**	2.09×	0.42×	**76.4**	1.23×	0.24×
GraNet$_{2.5\times}$ ($s_i = 0.5$) (ours)	**77.1**	0.85×	0.34×	**76.4**	0.49×	0.20×

We only run this experiment once due to the limited resources. We set $t_0 = 0$ and $t_f = 30$ for both GraNet ($s_i = 0$) and GraNet ($s_i = 0.5$) on ImageNet. Again, GraNet ($s_i = 0$) outperforms

GMP consistently with only half training FLOPs and achieves the highest accuracy among all the dense-to-sparse methods at sparsity of 0.9. Surprisingly, GraNet ($s_i = 0.5$) significantly boosts the sparse-to-sparse training performance, even over the dense-to-sparse training. Concretely, GraNet ($s_i = 0.5$) outperforms RigL by 0.9% and 1.5% at sparsity 0.8 and 0.9, respectively. To the best of our knowledge, this is the first time in the literature that sparse-to-sparse training reaches a test accuracy of 76% with ResNet-50 on ImageNet at sparsity 0.8, without extension of training time. It is reasonable for GraNet ($s_i = 0.5$) to achieve better accuracy than RigL, since the denser models at the beginning help GraNet explore more the parameter space. According to the In-Time Over-Parameterization hypothesis [37], the performance of sparse training methods is highly correlated with the total number of parameters that the sparse model has visited. We increase the training time of GraNet by 2.5× (250 epochs) and compare with the baselines with longer training time. With 50% fewer training time and 60% fewer training FLOPs, GraNet can match the performance of RigL trained with 500 epochs.

We further report the training/inference FLOPs required by all pruning methods. Compared with other dense-to-sparse methods, the final networks learned by GraNet ($s_i = 0$) require more FLOPs to test, whereas the overall training FLOPs required by GraNet ($s_i = 0$) are smaller than others. Even though starting from a denser model, GraNet ($s_i = 0.5$) requires less training and inference FLOPs than the state-of-the-art method, i.e., RigL. The sparsity budgets for 0.9 sparse ResNet-50 on ImageNet-1K learned by our methods are reported in Appendix D. We also report how FLOPs of the pruned ResNet-50 evolve during the course of training in Appendix E.

4.4 Effect of the Initial Sparsity

As we mentioned earlier, the denser initial network is the key factor in the success of GraNet. We conducted experiments to study the effect of the initial sparsity on GraNet with ResNet-50 on ImageNet. The initial sparsity is chosen from [0.0, 0.5, 0.6, 0.7, 0.8, 0.9] and the final sparsity is fixed as 0.9. The results are shared in Table 4. We can see the training FLOPs of GraNet are quite robust to the initial sparsity. Surprisingly yet reasonably, it seems that the the smaller the initial sparsity is (up to 0.5), the better final sparsity distribution GraNet finds, with higher test accuracy and fewer feedforward FLOPs. The lower feedforward FLOPs of the final network perfectly balance the overhead caused by the denser initial network.

Table 4: Effect of the initial sparsity on GraNet with ResNet-50 on ImageNet. The training/test FLOPs are normalized with the FLOPs of a dense model.

Method	s_i	s_f	Top-1 [%] Accuracy	FLOPs (Train)	FLOPs (Test)
GraNet	0.0	0.9	74.2	0.23×	0.16×
GraNet	0.5	0.9	74.5	0.25×	0.20×
GraNet	0.6	0.9	74.4	0.25×	0.22×
GraNet	0.7	0.9	74.2	0.24×	0.22×
GraNet	0.8	0.9	74.1	0.25×	0.24×
RigL	0.9	0.9	73.0	0.25×	0.24×

4.5 Performance of GraNet at Extreme Sparsities

In this section, we share the results of GraNet and RigL at extreme sparsities. The initial sparsity is set as 0.5. When the final sparsity is relatively smaller (e.g., 0.8, 0.9), GraNet requires a lower (or the same) number of training FLOPs than RigL, whereas GraNet requires more training FLOPs than RigL when the final sparsity is extremely high (e.g., 0.95, 0.965). This makes sense since when the sparsity is extremely high, the saved FLOPs count of the distribution discovered by GraNet is too small to amortize the overhead caused by denser initial models. Yet, the increased number of training FLOPs of GraNet leads to substantial accuracy improvement (> 2%) over RigL. The efficiency of GraNet ($s_i = 0.5$) comes from two important technical differences compared with RigL: (1) better final sparse distribution discovered by global pruning; (2) a shorter period of gradual pruning time (the first 30 epochs for ResNet-50 on ImageNet). Although starting with more parameters, the global pruning enables GraNet to quickly (first 30 epochs) evolve to a better sparsity distribution with lower test FLOPs than ERK. After 30 epochs of gradual pruning, the network continues to be trained with

Table 5: Comparison between GraNet and RigL at extreme sparsities with ResNet-50 on ImageNet. The training/test FLOPs are normalized with the FLOPs of a dense model.

Method	s_i	s_f	Top-1 [%] Accuracy	FLOPs (Train)	FLOPs (Test)
RigL	0.8	0.8	75.1	0.42×	0.42×
GraNet	0.5	0.8	76.0	0.37×	0.35×
RigL	0.9	0.9	73.0	0.25×	0.24×
GraNet	0.5	0.9	74.5	0.25×	0.20×
RigL	0.95	0.95	69.7	0.12×	0.12×
GraNet	0.5	0.95	72.3	0.17×	0.12×
RigL	0.965	0.965	67.2	0.11×	0.11×
GraNet	0.5	0.965	70.5	0.15×	0.09×

this better distribution for 70 epochs, so that the overhead in the early training phase with larger training FLOPs is amortized by the later and longer training phase with fewer training FLOPs.

4.6 Ablation Study of Random Reinitialization

Next, we ask whether what GraNet learned are the specific sparse connectivity or the sparse connectivity together with the weight values. We randomly reinitialize the pruned network with the same mask and retrain it. The results are given in Figure 4. The performance of the reinitialized networks falls significantly short of the performance achieved by GraNet ($s_i = 0$), indicating that what was learned by GraNet is the sparse connectivity together with the weight values. Besides, we find that the retraining performance of GraNet is higher than GMP. This further confirms that Zero-Cost Neuroregeneration helps the gradual pruning find more accurate mask positions.

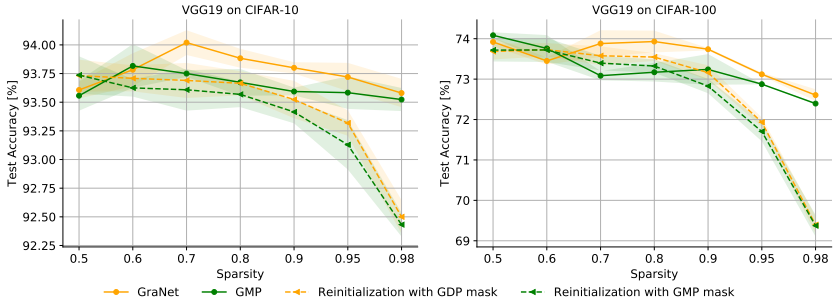

Figure 4: Reinitialization ablation on subnetworks discovered by GMP and GraNet ($s_i = 0$).

4.7 Comparison between Re-training and Extended Training

In this section, we study if re-training techniques can further improve the performance of the subnetworks discovered by GraNet. The authors of Lottery Ticket Hypothesis (LTH) [10] introduced a retraining technique, even if they did not evaluate it as such, where the subnetworks discovered by iterative magnitude pruning can be re-trained in isolation to full accuracy with the original initializations. Later on, learning rate rewinding (LRR) [54] was proposed further to improve the re-training performance by only rewinding the learning rate. Since GraNet also utilizes magnitude pruning to discover subnetworks, it is natural to test if these re-training techniques can bring benefits to GraNet. As shown in Table 6, both re-training techniques do not bring benefits to GraNet. Instead of re-training the subnetworks, we find that simply extending the training time significantly boosts the performance of GraNet with similar computational costs.

Table 6: Effects of LTH and LRR on the subnetworks learned by GraNet. Methods with $_2\times$ refer to extending the training steps by 2 times. The results are reported with top-1 test accuracy [%].

Dataset	CIFAR-10			CIFAR-100		
Pruning ratio	90%	95%	98%	90%	95%	98%
VGG-19 (Dense)	93.85±0.05	-	-	73.43±0.08	-	-
GraNet ($s_i = 0$)	93.80±0.10	93.72±0.11	93.63±0.08	73.74±0.30	73.10±0.04	72.35±0.26
+ Lottery Ticket Hypothesis	93.63±0.04	93.29±0.05	92.46±0.08	72.97±0.25	71.76±0.22	69.28±0.36
+ Learning Rate Rewinding	93.84±0.14	93.72±0.06	93.53±0.04	73.71±0.08	73.24±0.24	72.50±0.26
GraNet$_{2\times}$ ($s_i = 0$)	**94.17±0.03**	**93.98±0.07**	**93.94±0.11**	**74.80±0.29**	**73.65±0.32**	**73.63±0.05**
ResNet-50 (Dense)	94.75±0.01	-	-	78.23±0.18	-	-
GraNet ($s_i = 0$)	94.49±0.08	94.44±0.01	94.34±0.17	77.29±0.45	76.71±0.26	76.10±0.20
+ Lottery Ticket Hypothesis	93.96±0.10	93.70±0.15	92.94±0.14	75.74±0.19	74.31±0.10	71.99±0.08
+ Learning Rate Rewinding	94.55±0.13	94.39±0.13	94.20±0.25	77.40±0.14	76.90±0.19	75.75±0.25
GraNet$_{2\times}$ ($s_i = 0$)	**95.09±0.15**	**94.84±0.11**	**94.69±0.24**	**78.18±0.20**	**78.17±0.20**	**77.15±0.29**

5 Conclusion, and Reflection of Broader Impacts

In this paper, we re-emphasize the merit of during-training pruning. Compared with the recently proposed works, i.e., LTH and SNIP, during-training pruning is an efficient yet performant class of pruning methods that have received much less attention. We quantitatively study pruning during training from the perspective of pruning plasticity. Inspired by the findings from pruning plasticity and the mechanism of neuroregeneration in the nervous system, we further proposed a novel sparse

training method, GraNet, that performs the cost-free connection regeneration during training. GraNet advances the state of the art in both dense-to-sparse training and sparse-to-sparse training.

Our paper re-emphasizes the great potential of during-training pruning in reducing the training/inference resources required by ML models without sacrificing accuracy. It has a significant environmental impact on reducing the energy cost of the ML models and CO2 emissions [1, 53, 15, 56, 62].

6 Acknowledgement

This project is partially financed by the Dutch Research Council (NWO). We thank the reviewers for the constructive comments and questions, which improved the quality of our paper.

References

[1] Z. Atashgahi, G. Sokar, T. van der Lee, E. Mocanu, D. C. Mocanu, R. Veldhuis, and M. Pechenizkiy. Quick and robust feature selection: the strength of energy-efficient sparse training for autoencoders. *arXiv:2012.00560*, 2020.

[2] B. Bartoldson, A. Morcos, A. Barbu, and G. Erlebacher. The generalization-stability tradeoff in neural network pruning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20852–20864. Curran Associates, Inc., 2020.

[3] G. Bellec, D. Kappel, W. Maass, and R. Legenstein. Deep rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*, 2018. arXiv:1711.05136 (2017).

[4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[5] T. Chen, J. Frankle, S. Chang, S. Liu, Y. Zhang, M. Carbin, and Z. Wang. The lottery tickets hypothesis for supervised and self-supervised pre-training in computer vision models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16306–16316, June 2021.

[6] P. de Jorge, A. Sanyal, H. Behl, P. Torr, G. Rogez, and P. K. Dokania. Progressive skeletonization: Trimming more fat from a network at initialization. In *International Conference on Learning Representations*, 2021. arXiv:cs.CV/2006.09081.

[7] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. *Twenty-eighth Conference on Neural Information Processing Systems. arXiv:1404.0736*, 2014.

[8] T. Dettmers and L. Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

[9] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pages 2943–2952. PMLR, 2020.

[10] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. arXiv:1803.03635.

[11] J. Frankle, G. K. Dziugaite, D. Roy, and M. Carbin. Pruning neural networks at initialization: Why are we missing the mark? In *International Conference on Learning Representations*, 2021. arXiv:2009.08576.

[12] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin. The lottery ticket hypothesis at scale. *arXiv preprint arXiv:1903.01611*, 2019.

[13] T. Gale, E. Elsen, and S. Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[14] T. Gale, M. Zaharia, C. Young, and E. Elsen. Sparse gpu kernels for deep learning. *arXiv preprint arXiv:2006.10901*, 2020.

[15] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn. Estimation of energy consumption in machine learning. *Journal of Parallel and Distributed Computing*, 134:75–88, 2019.

[16] T. Gebhart, U. Saxena, and P. Schrater. A unified paths perspective for pruning at initialization. *arXiv preprint arXiv:2101.10552*, 2021.

[17] J. Guo, Y. Li, W. Lin, Y. Chen, and J. Li. Network decoupling: From regular to depthwise separable convolutions. *arXiv preprint arXiv:1808.05517*, 2018.

[18] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[19] B. Hassibi and D. G. Stork. *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann, 1993.

[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[21] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021.

[22] I. Hubara, B. Chmiel, M. Island, R. Banner, S. Naor, and D. Soudry. Accelerated sparse neural training: A provable and efficient method to find n: M transposable masks. *arXiv preprint arXiv:2102.08124*, 2021.

[23] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[24] S. A. Janowsky. Pruning versus clipping in neural networks. *Physical Review A*, 39(12):6600, 1989.

[25] S. Jayakumar, R. Pascanu, J. Rae, S. Osindero, and E. Elsen. Top-kast: Top-k always sparse training. *Advances in Neural Information Processing Systems*, 33:20744–20754, 2020.

[26] E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. Siegelbaum, A. J. Hudspeth, and S. Mack. *Principles of neural science*, volume 4. McGraw-hill New York, 2000.

[27] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[28] A. Kusupati, V. Ramanujan, R. Somani, M. Wortsman, P. Jain, S. Kakade, and A. Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *International Conference on Machine Learning*, pages 5544–5555. PMLR, 2020.

[29] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[30] N. Lee, T. Ajanthan, S. Gould, and P. H. S. Torr. A signal propagation perspective for pruning neural networks at initialization. In *International Conference on Learning Representations*, 2020. arXiv:1906.06307.

[31] N. Lee, T. Ajanthan, and P. H. Torr. Snip: Single-shot network pruning based on connection sensitivity. *International Conference on Learning Representations*, 2018.

[32] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *International Conference on Learning Representations*, 2016.

[33] T. Lin, S. U. Stich, L. Barba, D. Dmitriev, and M. Jaggi. Dynamic model pruning with feedback. In *International Conference on Learning Representations*, 2020.

[34] J. Liu, Z. Xu, R. Shi, R. C. C. Cheung, and H. K. So. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. In *International Conference on Learning Representations*, 2020.

[35] S. Liu, D. C. Mocanu, A. R. R. Matavalam, Y. Pei, and M. Pechenizkiy. Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. *Neural Computing and Applications*, 33(7):2589–2604, 2021.

[36] S. Liu, D. C. Mocanu, Y. Pei, and M. Pechenizkiy. Selfish sparse rnn training. In *Proceedings of the 39th International Conference on Machine Learning*, pages 6893–6904. PMLR, 2021.

[37] S. Liu, L. Yin, D. C. Mocanu, and M. Pechenizkiy. Do we actually need dense over-parameterization? in-time over-parameterization in sparse training. In *Proceedings of the 39th International Conference on Machine Learning*, pages 6989–7000. PMLR, 2021.

[38] T. Liu and F. Zenke. Finding trainable sparse networks through neural tangent transfer. In *International Conference on Machine Learning*, pages 6336–6347. PMLR, 2020.

[39] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. *International Conference on Learning Representations*, 2019.

[40] C. Louizos, M. Welling, and D. P. Kingma. Learning sparse neural networks through l_0 regularization. *International Conference on Learning Representations*, 2018.

[41] M. Mahar and V. Cavalli. Intrinsic mechanisms of neuronal axon regeneration. *Nature Reviews Neuroscience*, 19(6):323–337, 2018.

[42] D. Mittal, S. Bhardwaj, M. M. Khapra, and B. Ravindran. Recovering from random pruning: On the plasticity of deep convolutional neural networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 848–857. IEEE, 2018.

[43] D. C. Mocanu, E. Mocanu, T. Pinto, S. Curci, P. H. Nguyen, M. Gibescu, D. Ernst, and Z. A. Vale. Sparse training theory for scalable and efficient agents. *International Conference on Autonomous Agents and Multiagent Systems (AAMAS). arXiv:2103.01636*, 2021.

[44] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):2383, 2018.

[45] D. Molchanov, A. Ashukha, and D. Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pages 2498–2507. PMLR, 2017.

[46] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

[47] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. *International Conference on Learning Representations*, 2016.

[48] H. Mostafa and X. Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. *International Conference on Machine Learning*, 2019.

[49] M. C. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in neural information processing systems*, pages 107–115, 1989.

[50] M. C. Mozer and P. Smolensky. Using relevance to reduce network size automatically. *Connection Science*, 1(1):3–16, 1989.

[51] P. G. Nagappan, H. Chen, and D.-Y. Wang. Neuroregeneration and plasticity: a review of the physiological mechanisms for achieving functional recovery postinjury. *Military Medical Research*, 7(1):1–16, 2020.

[52] Nvidia. Nvidia a100 tensor core gpu architecture. *https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf*, 2020.

[53] D. Patterson, J. Gonzalez, Q. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. So, M. Texier, and J. Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

[54] A. Renda, J. Frankle, and M. Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *International Conference on Learning Representations*, 2020. arXiv:2003.02389.

[55] P. Savarese, H. Silva, and M. Maire. Winning the lottery with continuous sparsification. *arXiv preprint arXiv:1912.04427*, 2019.

[56] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni. Green ai. *arXiv preprint arXiv:1907.10597*, 2019.

[57] Y. Shen, L. Shen, H.-Z. Huang, X. Wang, and W. Liu. Cpot: Channel pruning via optimal transport. *arXiv preprint arXiv:2005.10451*, 2020.

[58] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*, 2014.

[59] S. P. Singh and D. Alistarh. Woodfisher: Efficient second-order approximation for neural network compression. *Advances in Neural Information Processing Systems*, 33, 2020.

[60] S. Srinivas, A. Subramanya, and R. Venkatesh Babu. Training sparse neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 138–145, 2017.

[61] N. Strom. Scalable distributed dnn training using commodity gpu cloud computing. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[62] E. Strubell, A. Ganesh, and A. McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

[63] H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems. arXiv:2006.05467*, 2020.

[64] H. Touvron, A. Vedaldi, M. Douze, and H. Jegou. Fixing the train-test resolution discrepancy. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[65] S. Verdenius, M. Stol, and P. Forré. Pruning via iterative ranking of sensitivity statistics. *arXiv preprint arXiv:2006.00896*, 2020.

[66] S. Verma and J.-C. Pesquet. Sparsifying networks via subdifferential inclusion. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10542–10552. PMLR, 18–24 Jul 2021.

[67] C. Wang, G. Zhang, and R. Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020.

[68] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[69] M. Wortsman, A. Farhadi, and M. Rastegari. Discovering neural wirings. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[70] X. Xiao, Z. Wang, and S. Rajasekaran. Autoprune: Automatic network pruning by regularizing auxiliary parameters. *Advances in neural information processing systems*, 32, 2019.

[71] Y. Xu, Y. Li, S. Zhang, W. Wen, B. Wang, Y. Qi, Y. Chen, W. Lin, and H. Xiong. Trained rank pruning for efficient deep neural networks. *arXiv preprint arXiv:1812.02402*, 2018.

[72] D. Yang, A. Ghasemazar, X. Ren, M. Golub, G. Lemieux, and M. Lis. Procrustes: a dataflow and accelerator for sparse deep neural network training. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 711–724. IEEE, 2020.

[73] G. Yiu and Z. He. Glial inhibition of cns axon regeneration. *Nature Reviews Neuroscience*, 7(8):617–627, 2006.

[74] H. You, C. Li, P. Xu, Y. Fu, Y. Wang, X. Chen, R. G. Baraniuk, Z. Wang, and Y. Lin. Drawing early-bird tickets: Towards more efficient training of deep networks. *arXiv preprint arXiv:1909.11957*, 2019.

[75] G. Yuan, L. Shen, and W.-S. Zheng. A block decomposition algorithm for sparse optimization. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 275–285, 2020.

[76] A. Zhou, Y. Ma, J. Zhu, J. Liu, Z. Zhang, K. Yuan, W. Sun, and H. Li. Learning n:m fine-grained structured sparse neural networks from scratch. In *International Conference on Learning Representations*, 2021.

[77] M. H. Zhu and S. Gupta. To prune, or not to prune: Exploring the efficacy of pruning for model compression, 2018.

A Remaining Experimental Results of Pruning Plasticity

We also studied pruning plasticity on structured pruning. In particular, we choose the filter pruning method used in Li et al. [32]. The pruning criterion is the absolute weight sum of each nonzero filter and the regeneration criterion is the absolute gradient sum of each zero filter. We first pre-train four sets of neural networks from scratch with various structured sparsity, including 0, 0.10, 0.50, and 0.70, noted as "Pre-trained Sparsity" in the figure title. To measure the plasticity of these pre-trained models, we choose four different pruning rates noted as "Pruning rate" to remove filters from these pre-trained models. The results of ResNet-20 and VGG-19 are shown as below.

A.1 ResNet-20 on CIFAR-10 with Structured Filter Pruning

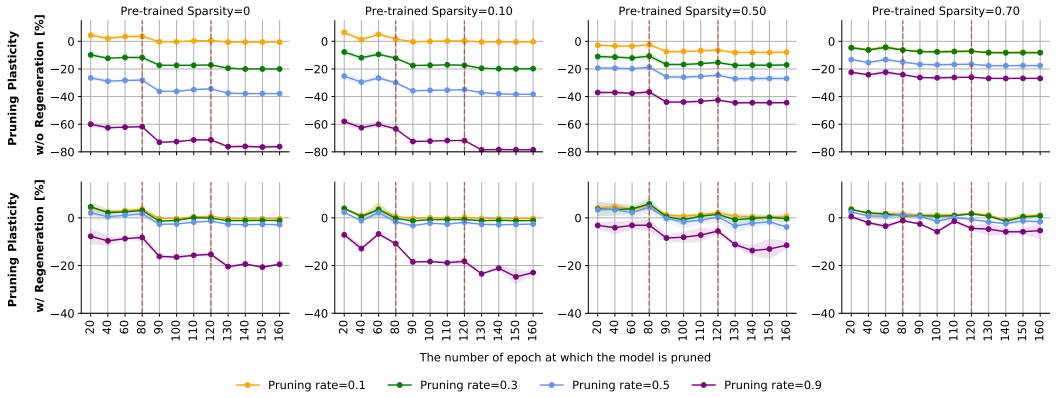

Figure 5: **Structured Pruning:** Pruning plasticity of under a 30-epochs continued training with and without connection regeneration for ResNet-20 on CIFAR-10. The vertical red lines refer to the points when the learning rate is decayed. The pruning method is uniform pruning.

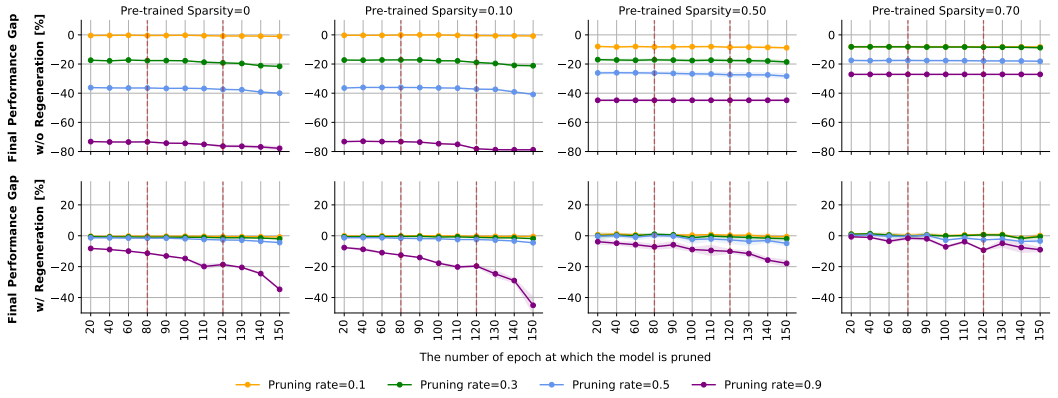

Figure 6: **Structured Pruning:** Final performance gap between the unpruned models and the pruned models for ResNet-20 on CIFAR-10. The vertical red lines refer to the points when the learning rate is decayed.

A.2 VGG-19 on CIFAR-10 with Structured Filter Pruning

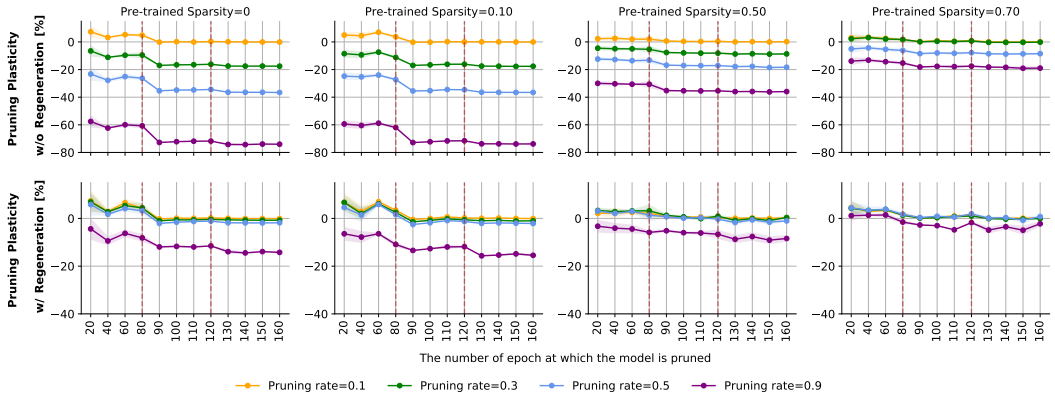

Figure 7: **Structured Pruning:** Pruning plasticity of under a 30-epochs continued training with and without connection regeneration for VGG-19 on CIFAR-10. The vertical red lines refer to the points when the learning rate is decayed. The pruning method is uniform pruning.

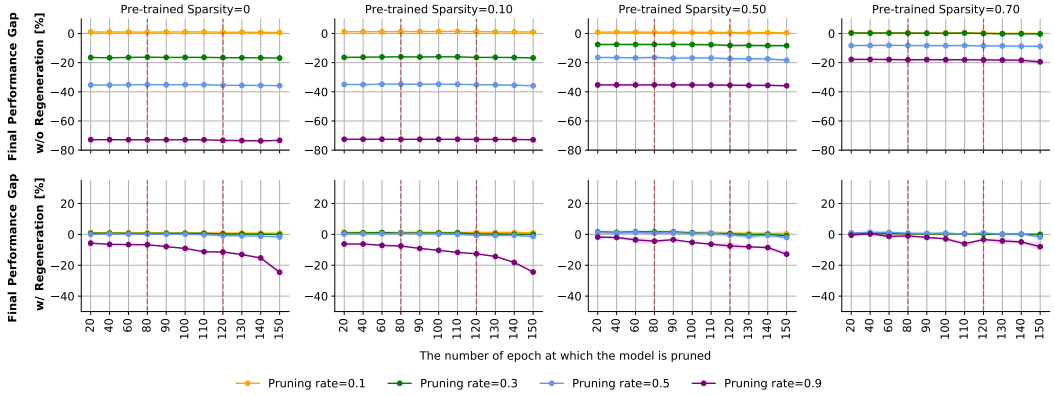

Figure 8: **Structured Pruning:** Final performance gap between the unpruned models and the pruned models for VGG-19 on CIFAR-10. The vertical red lines refer to the points when the learning rate is decayed.

A.3 ResNet-20 on CIFAR-10 with Unstructured Uniform Pruning

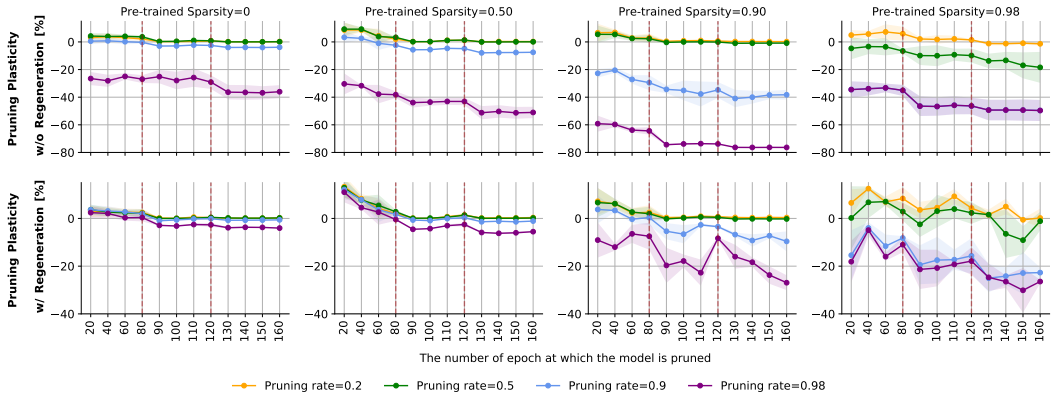

Figure 9: **Unstructured Pruning:** Pruning plasticity under a 30-epochs continued training with and without connection regeneration for ResNet-20 on CIFAR-10. The vertical red lines refer to the points when the learning rate is decayed.

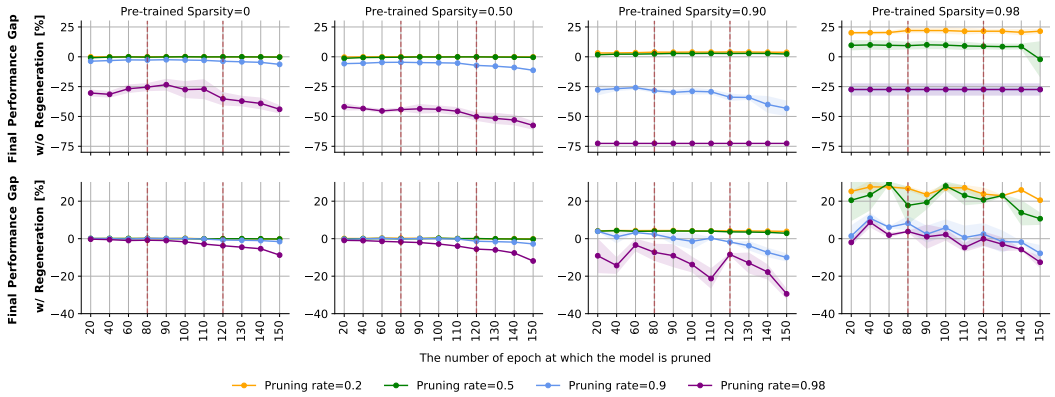

Figure 10: **Unstructured Pruning:** Final performance gap between the unpruned models and the pruned models for ResNet-20 on CIFAR-10. The vertical red lines refer to the points when the learning rate is decayed. The pruning method is uniform pruning.

A.4 VGG-19 on CIFAR-10 with Unstructured Global Pruning

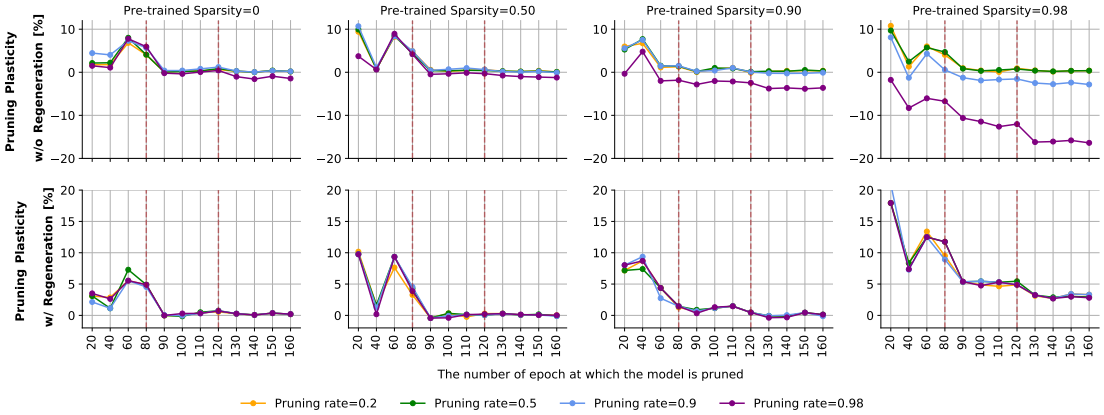

Figure 11: **Unstructured Pruning:** Pruning plasticity under a 30-epochs continued training with and without connection regeneration for VGG-19 on CIFAR-10. The vertical red lines refer to the points when the learning rate is decayed. The pruning method is global pruning.

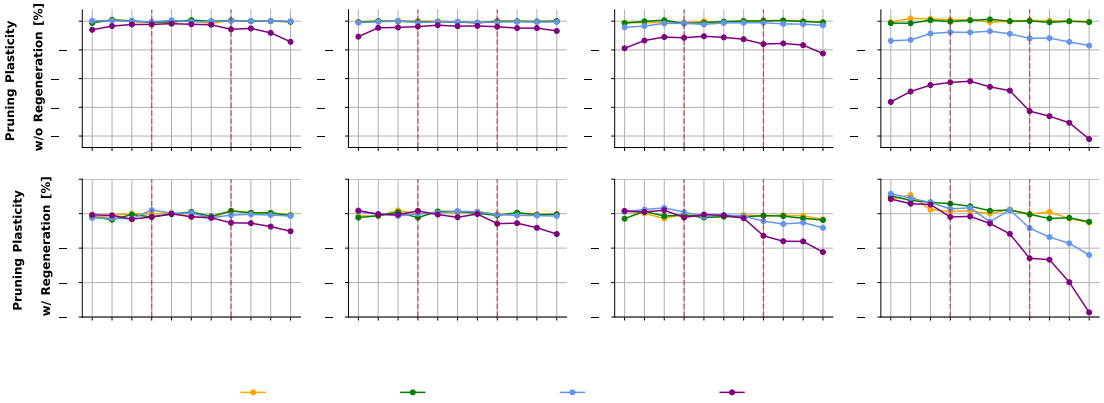

Figure 12: **Unstructured Pruning:** Final performance gap between the unpruned models and the pruned models for VGG-19 on CIFAR-10. The vertical red lines refer to the points when the learning rate is decayed. The pruning method is global pruning.

A.5 VGG-19 on CIFAR-10 with Unstructured Uniform Pruning

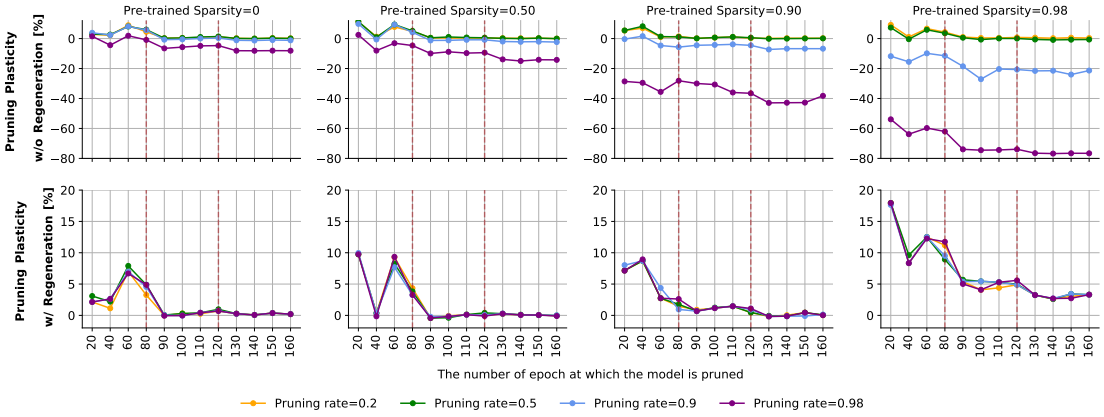

Figure 13: **Unstructured Pruning:** Pruning plasticity under a 30-epochs continued training with and without connection regeneration for VGG-19 on CIFAR-10. The vertical red lines refer to the points when the learning rate is decayed. The pruning method is uniform pruning.

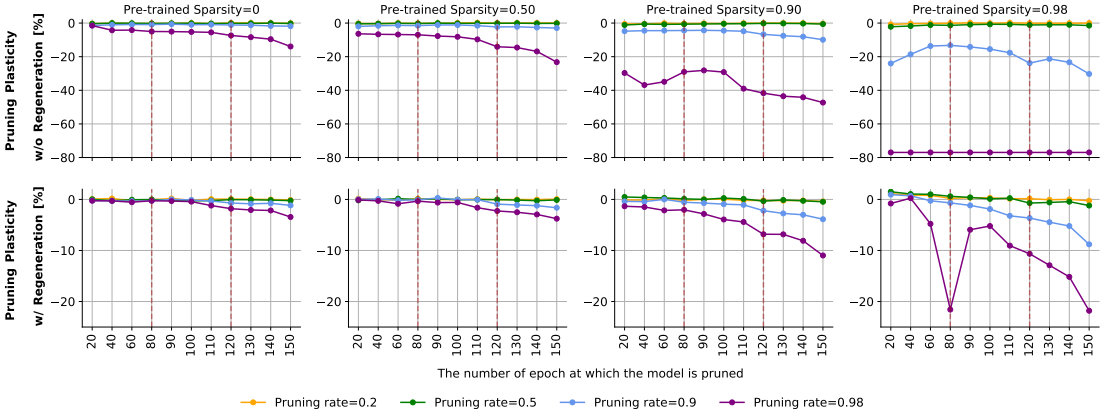

Figure 14: **Unstructured Pruning:** Final performance gap between the unpruned models and the pruned models for VGG-19 on CIFAR-10. The vertical red lines refer to the points when the learning rate is decayed. The pruning method is uniform pruning.

B Implementation Details of GraNet

In this appendix, we share in detail the pseudocode and implementation details of GraNet.

B.1 Algorithm

The pseudocode of GraNet is shared in Algorithm 1. The only difference between sparse-to-sparse training and dense-to-sparse training is the choices of initial sparsity s_i. For dense-to-sparse training, we need to set the initial sparsity of the model $s_i = 0$. To perform sparse-to-sparse training, we need to make sure the model is sparse at the beginning by setting the initial sparsity larger than 0, i.e., $s_i > 0$.

Algorithm 1 The pseudocode of GraNet.

Require: Model weights $W \in \mathbb{R}^d$, initial sparsity s_i, target sparsity s_f, gradual pruning starting point t_0, gradual pruning end point t_f, gradual pruning frequency ΔT.

$W \leftarrow$ randomly initialize W with initial sparsity s_i
for each training step t **do**
  training $W \leftarrow \text{SGD}(W)$
  if $t_o \leq t \leq t_f$ **and** $(t \bmod \Delta \text{T}) == 0$ **then**
    gradual pruning with the pruning rate produced by Eq. 1
    zero-cost neuroregeneration with Eq. 2 and Eq. 3
  end if
end for

B.2 Hyperparameters

We share the hyperparameter choices in our experiments in Table 7.

Table 7: Experiment hyperparameters of GraNet used in this paper. Learning Rate (LR), Batch Size (BS), Epochs, Learning Rate Drop (LR Drop), Weight Decay (WD), Sparse Initialization (Sparse Init), Gradual Pruning Frequency (ΔT), Initial Sparsity (s_i), Starting Epoch of Gradual Pruning (t_0), End Epoch of Gradual Pruning (t_f), Initial Neuroregeneration Ratio (r), Neuroregeneration Ratio (r Sche), etc.

Model	Data	Methods	LR	BS	Epochs	LR Drop, Epochs	WD	Sparse Init	Gradual Pruning				Neuroregeneration	
									ΔT	s_i	t_0	t_f	r	r Sche
VGG-19	CIFAR-10/100	dense-to-sparse	0.1	128	160	10x, [80, 120]	5e-4	Dense	1000	0	0 epoch	110 epoch	0.5	Cosine
	CIFAR-10/100	sparse-to-sparse	0.1	128	160	10x, [80, 120]	5e-4	ERK	1000	0.5	0 epoch	80 epoch	0.5	Cosine
ResNet-50	CIFAR-10/100	dense-to-sparse	0.1	128	160	10x, [80, 120]	5e-4	Dense	1000	0	0 epoch	110 epoch	0.5	Cosine
	CIFAR-10/100	sparse-to-sparse	0.1	128	160	10x, [80, 120]	5e-4	ERK	1000	0.5	0 epoch	80 epoch	0.5	Cosine
	ImageNet	dense-to-sparse	0.1	64	100	10x, [30, 60, 90]	1e-4	Dense	4000	0	0 epoch	30 epoch	0.5	Cosine
	ImageNet	sparse-to-sparse	0.1	64	100	10x, [30, 60, 90]	1e-4	ERK	4000	0.5	0 epoch	30 epoch	0.5	Cosine

B.3 Implementation

The implementation used in the paper is modified based on the open-source code of Sparse Momentum repository[2] introduced by [8]. We added VGG-19 with batchnorm from the GraSP repository[3]. The code for calculating the inference FLOPs of ResNet-50 on ImageNet is modified based on the open-source code provided in the rethinking-network-pruning repository[4]. For the training FLOPs, we follow the way of approximating the training FLOPs of RigL [9], where the FLOPs of the backward pass are around twice the ones of the forward pass.

[2] https://github.com/TimDettmers/sparse_learning

[3] https://github.com/alecwangcq/GraSP

[4] https://github.com/Eric-mingjie/rethinking-network-pruning/blob/master/imagenet/weight-level/compute_flops.py

C Implementation Details of GMP

In this appendix, we share in detail the pseudocode and implementation details of GMP.

C.1 Algorithm

Gradual Magnitude Pruning (GMP), introduced in [77] and studied further in [13], gradually sparsifies the neural network during the training process until the desired sparsity is reached. The pruning rate of each pruning iteration is:

$$s_t = s_f + (s_i - s_f)\left(1 - \frac{t - t_0}{n\Delta t}\right)^3 \; t \in \{t_0, t_0 + \Delta t, ..., t_0 + n\Delta t\} \tag{4}$$

The pseudocode of GMP is shown in Algorithm 2.

Algorithm 2 The pseudocode of GMP.

Require: Model weights $W \in \mathbb{R}^d$, initial sparsity s_i, target sparsity s_f, gradual pruning starting point t_0, gradual pruning end point t_f, gradual pruning frequency ΔT.

```
W ← randomly initialize W with initial sparsity s_i
for each training step t do
    training W ← SGD(W)
    if t_o ≤ t ≤ t_f and (t mod ΔT) == 0 then
        gradual pruning with the pruning rate produced by Eq. 1
    end if
end for
```

C.2 Hyperparameters

To demonstrate the effectiveness of Zero-Cost Neuroregeneration, we reproduce GMP with our implementation for CIFAR-10/100 so that the only difference between GMP and GraNet is the Zero-Cost Neuroregeneration. Hence, all the hyperparameters of GMP on CIFAR-10/100 are the same as GraNet.

It is surprising that the number of training FLOPs required by GraNet is smaller than GMP reported in [13]. Since the authors of [13] did not share the specific hyperparameters that used to produce the results of GMP, we guess the pruning of GMP happens late in training. Thus, it makes sense that GraNet requires fewer training FLOPs than GMP, as the dense training time of GraNet is much shorter than GMP.

C.3 Implementation

We reproduce GMP only for the results of CIFAR-10/100 for a fair comparison with GraNet. The results of GMP with ResNet-50 on ImageNet are obtained directly from [13].

We also test our implemented GMP with ResNet-50 on ImageNet. However, the performance is much worse than the results in [13] as shown below.

Table 8: Test accuracy of GMP ResNet-50 on ImageNet dataset with our own implementation.

Method	Top-1 Accuracy	FLOPs (Train)	FLOPs (Test)	TOP-1 Accuracy	FLOPs (Train)	FLOPs (Train)
Dense	76.8±0.09	1x (3.2e18)	1x (8.2e9)			
		S = 0.8			S = 0.9	
GMP [13]	75.6	0.56×	0.23×	73.9	0.51×	0.10×
GMP (our implementation)	74.6	0.34×	0.28×	73.3	0.23×	0.16×

We are aware that our GMP implementation has several differences from the original Tensorflow implementation used by [77, 13]. Firstly, since our implementation reset the weight values to zero once the weights are pruned, the pruned weights of GMP are also set to zero. However, in the original

GMP implementation, only the masks are set to zero and the weight values are kept, leading to a situation where the pruned weights can be regenerated back in a natural way. Secondly, the original GMP uses uniform pruning and keeps the first layer dense and the sparsity of the last layer no larger than 0.8. Same as GraNet, our implementation of GMP prune all the layers including the first layer and the last layer.

We also compare GMP and GraNet with uniform pruning as used in [13], as shown below. While the results with CIFAR-10 in unclear, GraNet outperforms GMP with CIFAR-100 consistently. As we expected, the performance using uniform pruning is generally worse than global pruning.

Table 9: Test accuracy of pruned VGG-19 and ResNet32 on CIFAR-10 and CIFAR-100 datasets using uniform pruning.

Dataset	CIFAR-10			CIFAR-100		
Pruning ratio	90%	95%	98%	90%	95%	98%
VGG-19 (Dense)	93.85±0.05	-	-	73.43±0.08	-	-
GMP [13]	**93.28±0.04**	92.76±0.10	91.64±0.18	71.88±0.29	71.02±0.27	66.16±0.23
GraNet ($s_i = 0$)	93.12±0.03	**92.88±0.08**	**91.87±0.06**	**72.37±0.01**	**71.48±0.25**	**70.14±0.18**
ResNet-50 (Dense)	94.75±0.01	-	-	78.23±0.18	-	-
GMP [13]	94.08±0.14	**94.20±0.24**	**93.66±0.44**	77.30±0.27	76.77±0.02	75.38±0.24
GraNet ($s_i = 0$)	**94.19±0.23**	94.16±0.26	93.64±0.25	**77.57±0.12**	**77.15±0.18**	**76.17±0.15**

D ResNet-50 Learnt Budgets and Backbone Sparsities

Table 10 summarizes the final sparsity budgets for 90% sparse ResNet50 on ImageNet-1K obtained by various methods. Backbone represents the sparsity budgets for all the CNN layers without the last fully-connected layer. VD refers to Variational Dropout [45] and GS refers to iterative magnitude pruning using a global threshold for global sparsity [18].

Table 10: ResNet-50 Learnt Budgets and Backbone Sparsities at Sparsity 0.9

Metric	Fully Dense Params	Fully Dense FLOPs	Sparsity (%)							
			GraNet ($s_i = 0$)	GraNet ($s_i = 0.5$)	STR	Uniform	ERK	SNFS	VD	GS
Overall	25502912	8178569216	89.99	89.98	90.23	90.00	90.07	90.06	90.27	89.54
Backbone	23454912	8174272512	89.89	90.65	92.47	90.00	89.82	89.44	91.41	90.95
Layer 1 - conv1	9408	118013952	53.50	40.60	59.80	90.00	58.00	2.50	31.39	35.11
Layer 2 - layer1.0.conv1	4096	236027904	54.60	43.40	83.28	90.00	0.00	2.50	39.50	56.05
Layer 3 - layer1.0.conv2	36864	231211008	78.80	64.50	89.48	90.00	82.00	2.50	67.87	75.04
Layer 4 - layer1.0.conv3	16384	102760448	78.00	67.40	85.80	90.00	4.00	2.50	64.87	70.31
Layer 5 - layer1.0.downsample.0	16384	102760448	79.30	72.90	83.34	90.00	4.00	2.50	60.38	66.88
Layer 6 - layer1.1.conv1	16384	102760448	76.60	67.30	89.89	90.00	4.00	2.50	61.35	75.09
Layer 7 - layer1.1.conv2	36864	231211008	76.70	62.10	90.60	90.00	82.00	2.50	64.38	80.42
Layer 8 - layer1.1.conv3	16384	102760448	74.10	54.50	91.70	90.00	4.00	2.50	65.83	80.00
Layer 9 - layer1.2.conv1	16384	102760448	72.20	58.80	88.07	90.00	4.00	2.50	68.75	75.21
Layer 10 - layer1.2.conv2	36864	231211008	72.70	58.50	87.03	90.00	82.00	2.50	70.86	74.95
Layer 11 - layer1.2.conv3	16384	102760448	73.20	57.30	90.99	90.00	4.00	2.50	54.05	79.28
Layer 12 - layer2.0.conv1	32768	205520896	68.30	49.30	85.95	90.00	43.00	2.50	57.10	70.89
Layer 13 - layer2.0.conv2	147456	231211008	77.50	69.10	93.91	90.00	91.00	62.90	78.65	85.39
Layer 14 - layer2.0.conv3	65536	102760448	71.70	61.10	93.13	90.00	52.00	11.00	85.49	83.54
Layer 15 - layer2.0.downsample.0	131072	205520896	90.30	86.80	94.96	90.00	71.00	66.10	79.96	88.36
Layer 16 - layer2.1.conv1	65536	102760448	85.20	83.00	95.31	90.00	52.00	32.60	72.07	88.25
Layer 17 - layer2.1.conv2	147456	231211008	85.30	81.10	91.50	90.00	91.00	61.60	84.41	85.37
Layer 18 - layer2.1.conv3	65536	102760448	80.00	68.60	93.66	90.00	52.00	20.80	79.19	86.53
Layer 19 - layer2.2.conv1	65536	102760448	82.60	80.70	94.61	90.00	52.00	29.10	73.94	86.40
Layer 20 - layer2.2.conv2	147456	231211008	83.20	82.40	94.86	90.00	91.00	63.90	78.48	88.29
Layer 21 - layer2.2.conv3	65536	102760448	79.30	76.40	93.38	90.00	52.00	22.90	78.09	85.87
Layer 22 - layer2.3.conv1	65536	102760448	81.10	77.10	93.26	90.00	52.00	27.60	78.66	84.87
Layer 23 - layer2.3.conv2	147456	231211008	82.10	83.40	93.21	90.00	91.00	65.30	84.38	87.14
Layer 24 - layer2.3.conv3	65536	102760448	82.40	77.30	94.14	90.00	52.00	25.70	82.07	86.84
Layer 25 - layer3.0.conv1	131072	205520896	72.80	61.00	88.85	90.00	71.00	48.70	66.56	78.40
Layer 26 - layer3.0.conv2	589824	231211008	84.60	83.30	96.14	90.00	96.00	90.20	87.92	92.93
Layer 27 - layer3.0.conv3	262144	102760448	78.00	69.70	93.19	90.00	76.00	73.30	92.19	86.19
Layer 28 - layer3.0.downsample.0	524288	205520896	95.30	95.20	97.20	90.00	86.00	93.70	88.76	94.66
Layer 29 - layer3.1.conv1	262144	102760448	91.30	91.40	95.36	90.00	76.00	81.10	91.79	93.60
Layer 30 - layer3.1.conv2	589824	231211008	91.10	93.10	95.06	90.00	96.00	90.40	92.47	93.07
Layer 31 - layer3.1.conv3	262144	102760448	85.10	81.50	94.84	90.00	76.00	78.10	88.88	90.54
Layer 32 - layer3.2.conv1	262144	102760448	90.10	89.70	96.77	90.00	76.00	80.40	84.86	93.44
Layer 33 - layer3.2.conv2	589824	231211008	90.10	93.40	95.59	90.00	96.00	90.80	91.50	93.73
Layer 34 - layer3.2.conv3	262144	102760448	86.70	83.90	94.99	90.00	76.00	79.30	81.59	91.13
Layer 35 - layer3.3.conv1	262144	102760448	89.20	91.00	96.08	90.00	76.00	80.70	76.64	93.18
Layer 36 - layer3.3.conv2	589824	231211008	90.90	94.20	96.10	90.00	96.00	90.70	91.26	93.63
Layer 37 - layer3.3.conv3	262144	102760448	88.50	87.50	94.94	90.00	76.00	79.00	85.46	91.63
Layer 38 - layer3.4.conv1	262144	102760448	88.90	89.60	95.49	90.00	76.00	79.40	85.33	91.98
Layer 39 - layer3.4.conv2	589824	231211008	92.20	94.70	95.66	90.00	96.00	91.00	91.57	94.21
Layer 40 - layer3.4.conv3	262144	102760448	90.30	88.60	94.49	90.00	76.00	79.00	86.19	91.63
Layer 41 - layer3.5.conv1	262144	102760448	88.30	87.50	95.09	90.00	76.00	78.30	84.64	90.72
Layer 42 - layer3.5.conv2	589824	231211008	92.30	94.90	94.92	90.00	96.00	91.00	91.14	93.43
Layer 43 - layer3.5.conv3	262144	102760448	89.20	87.90	93.14	90.00	76.00	78.20	84.09	89.56
Layer 44 - layer4.0.conv1	524288	205520896	80.20	72.80	90.32	90.00	86.00	85.80	77.90	85.35
Layer 45 - layer4.0.conv2	2359296	231211008	89.80	93.60	95.66	90.00	98.00	97.60	96.53	95.07
Layer 46 - layer4.0.conv3	1048576	51380224	84.70	82.40	91.14	90.00	88.00	93.20	93.52	89.21
Layer 47 - layer4.0.downsample.0	2097152	205520896	99.00	99.20	96.79	90.00	93.00	98.80	93.80	96.72
Layer 48 - layer4.1.conv1	1048576	102760448	93.10	95.60	93.69	90.00	88.00	94.10	94.96	92.69
Layer 49 - layer4.1.conv2	2359296	231211008	93.60	97.30	93.98	90.00	98.00	97.70	97.76	93.85
Layer 50 - layer4.1.conv3	1048576	102760448	90.30	90.80	90.48	90.00	88.00	94.20	94.53	89.84
Layer 51 - layer4.2.conv1	1048576	205520896	87.30	87.10	87.57	90.00	88.00	93.60	94.19	85.91
Layer 52 - layer4.2.conv2	2359296	231211008	91.70	96.80	84.37	90.00	98.00	97.90	94.92	87.14
Layer 53 - layer4.2.conv3	1048576	102760448	85.00	83.40	80.29	90.00	88.00	94.50	89.64	80.65
Layer 54 - fc	2048000	4096000	91.30	82.40	64.50	90.00	93.00	97.10	77.17	73.43

E FLOPs Dynamics During Training with ResNet-50 on ImageNet

To have an overview of how the FLOPs of the pruned model evolves during training, we share the FLOPs dynamics (inference on single sample) of the pruned ResNet-50 during the course of training in Figure 15. While starting from a model with a higher number of FLOPs compared with GraNet ($s_i = 0.5$), GraNet ($s_i = 0$) is gradually sparsified towards a sparse structure with lower FLOPs.

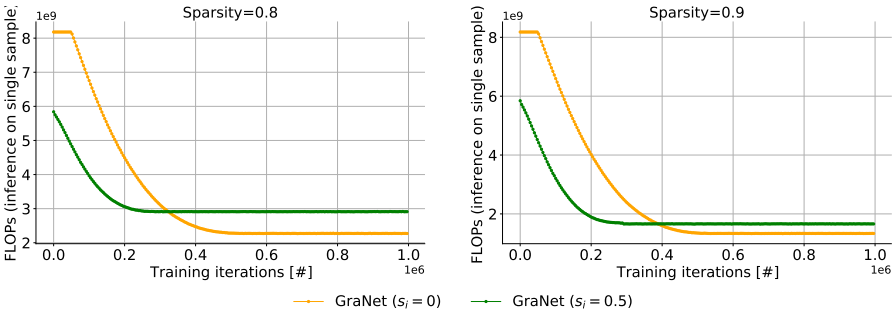

Figure 15: FLOPs dynamics (inference on single sample) of GraNet with ResNet-50 on ImageNet during training.