SPENDING YOUR WINNING LOTTERY BETTER AFTER DRAWING IT

Anonymous authors
Paper under double-blind review

ABSTRACT

Lottery Ticket Hypothesis (LTH) (Frankle & Carbin, 2019) suggests that a dense neural network contains a sparse sub-network that can match the performance of the original dense network when trained in isolation from scratch. Most works re-train the sparse sub-network with the same training protocols as its dense network, such as initialization, architecture blocks, and training recipes. However, till now it is unclear that whether these training protocols are optimal for sparse networks.

In this paper, we demonstrate that it is unnecessary for spare retraining to strictly inherit those properties from the dense network. Instead, by plugging in purposeful "tweaks" of the sparse subnetwork architecture or its training recipe, its retraining can be significantly improved than the default, especially at high sparsity levels. Combining all our proposed "tweaks" can yield the new state-of-the-art performance of LTH, and these modifications can be easily adapted to other sparse training algorithms in general. Specifically, we have achieved a significant and consistent performance gain of $1.05\% - 4.93\%$ for ResNet18 on CIFAR-100 over vanilla-LTH. Moreover, our methods are shown to generalize across datasets (CIFAR-10, CIFAR-100, TinyImageNet) and architectures (Vgg16, ResNet-18/ResNet-34, MobileNet). All codes will be publicly available.

1 INTRODUCTION

Deep neural networks (NN) have achieved significant progress in many tasks such as classification, detection, and segmentation. However, most existing models are computationally extensive and overparameterized, thus it is difficult to deploy these models in real-world devices. To address this issue, many efforts have been devoted to compressing the heavy model into a lightweight counterpart. Among them, network pruning (LeCun et al., 1990; Han et al., 2015a;b; Li et al., 2016; Liu et al., 2019), which identifies sparse sub-networks from the dense networks by removing unnecessary weights, stands as one of the most effective methods. Previous methods (Han et al., 2015b) usually prune the dense network after the standard training process to obtain the sparse sub-networks. The performance of the pruned network, however, decreases heavily as parts of the weights are removed, and retraining is thus required to recover the original performance (Han et al., 2015b).

The recent Lottery Ticket Hypothesis (LTH) (Frankle & Carbin, 2019) represents a major paradigm shift from conventional after-training pruning. LTH suggests that a dense NN contains sparse sub-networks, named "winning tickets", which can match the performance of the original dense NN when trained in isolation from scratch. Such a winning ticket can be "**drawn**" by finding the sparse weight mask, from dense training accompanied with iterative magnitude pruning (IMP). The found sparse mask is then applied to the original dense NN, and the masked sparse subnetwork is subsequently re-trained from scratch. Using a similar metaphor, We call the sparse re-training step as "**spending**" the lottery, after it is drawn.

In most (if not all) LTH literature (Frankle & Carbin, 2019; Frankle et al., 2019; Renda et al., 2020), the re-training step takes care of the masked subnetwork, which is re-trained with the same initialization (or rewinding) and same training recipe as its dense network. In plain language, "*You spend the same lottery ticket in the same way you draw it*". Recent evidence seems to support this convention by attributing LTH's success in recovering the original pruned solution Evci et al. (2020a). However, till now it is still unclear that whether the architecture blocks, initialization

regimes, or training recipes are necessarily optimal for the sparse network. Our question of curiosity is hence: "*Can you spend the same lottery ticket in a different yet better way than how you draw it*"?

Contrary to the common beliefs, this paper demonstrates that it is unnecessary for sparse network retraining ("spending the lottery") to stick to the protocol of dense network training or sparse mask finding ("drawing the lottery"). Instead, having sparse re-training purposely misaligned in some way from dense training can make the found sparse subnetwork work even better. Specifically, we investigate two possible aspects of modified sparse re-training:

- *Architecture tweaking:* after the sparse subnetwork is found, we modify the network architecture by: (a) injecting more residual skip-connections that are non-existent in dense networks; and (b) changing the ReLU neurons into smoother activation functions such as Swish (Ramachandran et al., 2017) and Mish (Misra, 2019).
- *Training recipe tweaking:* when training the (found or modified) sparse subnetwork architecture, we modify the training approach by: (c) changing the "lottery ticket initialization" by learned layer-wise scaling; and (d) replacing the one-hot labels with either naive or knowledge-distilled soft labels.

Each idea could be viewed as certain type of *learned smoothening* (we will explain later), and is plug-and-play in the sparse re-training stage of any LTH algorithm. Those techniques can be applied either alone or altogether, and can significantly boost the sparse re-training performance in large models and at high sparsities. Note that all above "tweaks" only affect the sparse re-training stage (we never alter the found sparse mask), but not the dense training/masking finding stage. In fact, our experiments will show that they boost sparse re-training much more than dense counterparts.

Our contributions can be summarized as:

- In contrast to the common wisdom that LTH sparse re-training needs to inherit (masked) network architecture, initialization, and training protocol from dense training, we for the first time demonstrate that purposely re-tweaking them will actually improve the sparse re-training step. That urges our rethinking of the LTH's true value.
- We investigate two groups of techniques to tweak the sparse subnetwork architecture and training recipe respectively. For the former, we inject new skip connections and replace new activation neurons. For the latter, we re-scale the initialization and soften the labels. Each of the techniques improves sparse re-training (much more than they can help dense counterparts), and altogether they lead to further boosts.
- Our extensive experimental results demonstrate that by plugging these techniques in LTH sparse retraining, we can significantly improve the performance of "winning tickets" at high sparsity levels and large models, setting the state-of-the-art performance bar of LTH. Furthermore, we show that they can benefit other sparse training algorithms in general, and provide visualizations to analyze their successes.

2 Background Work

2.1 The Lottery Ticket Hypothesis

The LTH (Frankle & Carbin, 2019) implies that initialization is the key for sparse network retraining. Beyond image classification (Liu et al., 2019; Savarese et al., 2020; Wang et al., 2020; You et al., 2020; Ma et al., 2021; Chen et al., 2020a), LTH has also been widely explored in numerous contexts, such as natural language processing (Gale et al., 2019; Yu et al., 2019; Prasanna et al., 2020; Chen et al., 2020b;c), reinforcement learning (Yu et al., 2019), self-supervised learning (Chen et al., 2020a), lifelong learning (Chen et al., 2021c), generative adversarial networks (Chen et al., 2021d; Kalibhat et al., 2020; Chen et al., 2021a), and graph neural networks (Chen et al., 2021b).

However, when retraining the sparse network, these works still strictly follow the same training recipe from dense networks. The most recent work (Tessera et al., 2021) reveals that focusing on initialization appears insufficient. Optimization strategies, regularization, and architecture choices also play significant roles in sparse network training. However, (Tessera et al., 2021) only compares sparse networks to their equivalent capacity dense networks, and most of their experiments are con-

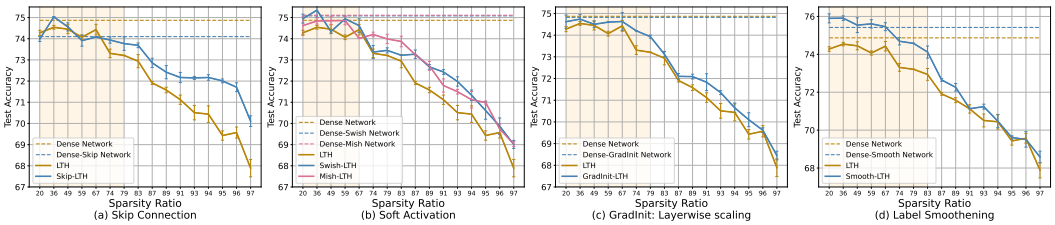


Figure 1: Results of testing accuracy for ResNet-18 on CIFAR-100 (three independent runs) for our architecture tweaking and training recipe tweaking techniques. X-axis denotes the sparsity level of the network. Dash straight lines denote the accuracy of dense network trained with and without proposed tweaking techniques. Performance gain by our techniques is more profound for sparse NN in comparison to dense NN. Shaded part of the graph illustrates the "Winning Tickets" identified by LTH with tweaking techniques. All experiments are conducted using *Iterative Magnitude Pruning* (IMP). See (Appendix A1.3) for more details.

ducted on multi-layer perceptron (MLP). Thus, it is unclear whether their conclusion can generalize to the ticket finding from dense CNNs.

2.2 Smoothness

Introducing smoothness into NNs, including on the weights, logits, or training trajectory, is a common techniques to improve the generalization and optimization (Jean & Wang, 1994). For labels, smoothness is usually introduced by replacing the hard target with soft labels (Szegedy et al., 2016) or soft logits (Hinton et al., 2015a). This uncertainty of labels helps to alleviate the overconfidence (Hein et al., 2019) and improves the generalization. Smoothness can also implemented by replacing the activation functions (Misra, 2019; Ramachandran et al., 2017), adding skip-connections in NNs (He et al., 2016), or averaging along the trajectory of gradient descent (Izmailov et al., 2018). These methods contribute to more stable gradient flows (Tessera et al., 2021) and smoother loss landscapes, but most of them have not been considered nor validated on sparse NNs.

3 Methodology

The LTH (Frankle & Carbin, 2019) suggests that the "winning tickets" can be found via the following three steps: (1) training a dense network from scratch; (2) pruning unnecessary weights to form the mask of a sparse sub-network; and (3) re-training the sub-network from the same initialization used in the dense model. For the third step, the retraining of sparse network usually inherits all properties, such as architecture blocks and training recipes, from dense networks. However, our experiments validate that those are not necessarily optimal for training sparse networks. To verify, we investigate two aspects of "tweaks" dedicated to the sparse re-training step: model architecture, and training recipe.

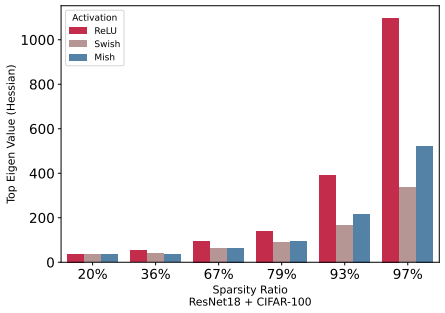
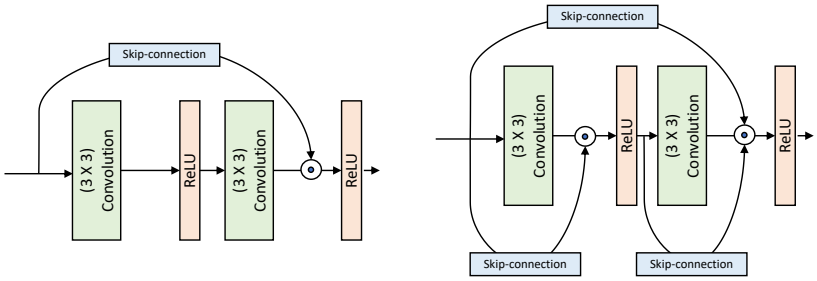


Figure 2: Comparison of the top eigenvalue of Hessian (mean across 10 random batches) of sparse NN trained with different activation functions. Comparatively smaller eigenvalues of soft-activation (Swish/Mish) based sparse NN wrt. ReLU based networks indicate the presence of high smoothness.

3.1 Model Architecture Tweaking

Replacing Smoother Activations Most deep NNs apply the Rectified Linear Units (ReLU) (Nair & Hinton, 2010) as the activation function. However, the gradient of ReLU changes suddenly around zero, and this non-smooth nature of ReLU is an obstacle to sparse retraining because it leads to high activation sparsity into the subnetwork, likely blocking the gradient flow (Appendix A1.1). To mitigate this issue and encourage healthier gradient flow, we replace the ReLU to Swish (Ramachandran et al., 2017) and Mish (Misra, 2019). Different from ReLU, Swish and Mish are both smooth non-monotonic activation functions. The non-monotonic property allows for the gradient of small negative inputs, which leads to a more stable gradient flow (Tessera et al., 2021). Meanwhile, the loss landscape of Swish and Mish are



(a) ResNet-18 Block (b) Modified ResNet-18 Block

Figure 3: Sketch of our modified ResNet-18 block to introduce additional skip-connections for sparse re-training.

proved to have smoother transition (Misra, 2019), which makes the loss function easier to optimize and hence makes the sparse network generalize better.

Injecting New Skip Connections High sparsity networks easily suffer from the layer-collapse (Tanaka et al., 2020), i.e., the premature pruning of an entire layer. This could make the sparse network untrainable, as the gradient cannot be backpropagated through that layer. The skip connection (or named "residual-addition") (He et al., 2016) was initially proposed to avoid gradient vanishing problem, and enables the training of a very deep model. It is later proven that skip connections can smooth the loss surfaces (Li et al., 2017b). That naturally motivates us to consider using more skip connections on the sparse networks to smoothen the landscape and preserve gradients, besides its possible mitigation effect when encountering collapsed layers.

Inspired by (Ding et al., 2021), we propose to "artificially" add new skip-connections during our sparse re-training. Figure 3 illustrates this architectural modifications to the traditional Resnet-18 block. Similar to existing residual connection in traditional ResNet-18 block, our newly introduced skip-connections add input of each (3×3) convolution block, to their output before the activation. The original motivation behind residual connections comes from their ability to allow gradient information to flow to earlier layers of the NN, thereby reducing the vanishing gradient problem during training (He et al., 2016). With high activation sparsity present in sparse subnetworks, additional skip-connections can facilitate healthy gradient flow and improve their trainability. Furthermore, (Li et al., 2017b) observed that with an increase in depth of networks, neural loss landscape becomes highly chaotic and leads to drop in generalization and trainability. They further observed that skip connections promote flatness and prevent transition to chaotic behaviour. Inspired by them, we added skip-connections to stabilize the initial chaotic training regime of sparse NN at high sparsity level and manage to prevent the transition to a sub-optimal behaviour.

3.2 Training recipe tweaking

Smoother Labels and Loss functions Label smoothing (Szegedy et al., 2016) has been widely used to improve the performance of dense networks trained with hard labels. Specifically, given the output probabilities p_k from the network and the target y_k, a network trained with hard labels aims to minimize the cross-entropy loss by $L_{\text{LS}} = -\sum_{k=1}^{K} y_k \log(p_k)$, where y_k is "1" for the correct class and "0" for others, and K is the number of classes. Label smoothing changes the target to a mixture of hard labels with a uniform distribution, and minimizes the cross-entropy between the modified target $_k^{LS}$ and output p_k. The modified target is defined as $y_k^{LS} = y_k(1-\alpha) + \alpha/K$, where α is the smooth ratio. This uniform distribution introduces smoothness into the training and encourages small logit gaps. Thus, label smoothing results in better model calibration and prevents overconfident predictions (Müller et al., 2019).

Meanwhile, recent works (Yuan et al., 2020b; Shen et al., 2021) suggest that Knowledge Distillation (KD) (Hinton et al., 2015a) is also one kind of label smoothing. KD aims to make (student) models learn from outputs produced by pretrained teacher models. In detail, given q_k^τ and p_k^τ, the outputs from both teacher and student after softmax with temperature τ, respectively, the network trained with KD aims to minimize $L_{\text{KD}} = \mathcal{KL}(q_k^\tau, p_k^\tau)$, where $\mathcal{KL}$ is the Kullback-Leibler divergence. Compared with uniform distribution, the outputs from teachers assign incorrect classes with different probabilities, which reveals a rich similarity structure over the data (Hinton et al., 2015a).

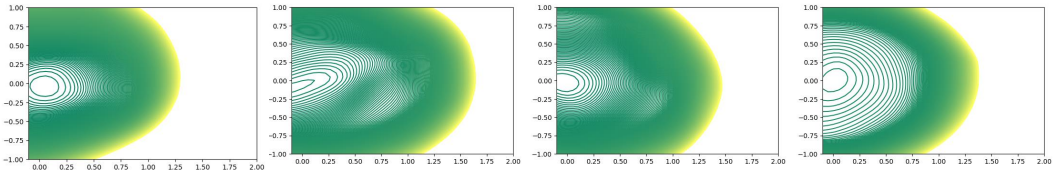


Figure 4: Comparison of loss surface contours of ResNet-18 models trained on CIFAR-100 with 91% sparsity in early training stage (epoch 5) using vanilla-LTH and our Smoothness-aware tweaks.

Both loss functions have been proved beneficial in dense network training (Yuan et al., 2020b). However, till now it is unclear whether they are still helpful in sparse network training. We demonstrate that their smoothness property can bring additional benefits for sparse network training, as they can flatten the loss landscapes and facilitate the training. We also observe that naive label smoothening works sufficiently well compared to KD: please refer to Section 4.6.

Layer-wise Re-scaled Initialization Carefully crafted initialization techniques that can prevent gradient explosion/vanishing in backpropagation have been an important part of the early success of feed-forward neural networks (He et al., 2016; Glorot & Bengio, 2010). Even with recent cleverly designed initialization rules, complex models with many layers and branches suffer from instability. For example, the Post-LN Transformer (Vaswani et al., 2017) can not converge without learning rate warmup using the default initialization.

In sparse subnetwork re-training, most of the existing works use common initialization methods (Glorot & Bengio, 2010; He et al., 2016) derived from dense NN (with the sparse mask applied). These initialization techniques ensure that the output distribution of every neuron in a layer is of zero-mean and unit variance by sampling a Gaussian distribution with a variance based on the number of incoming/outgoing connections for all the neurons in a dense layer. However, after sparsification, the number of incoming/outgoing connections is not identical for all the neurons in the layer (Evci et al., 2020b) and this raises direct concerns against the blind usage of dense network initialization for sparse subnetworks. Furthermore, (Evci et al., 2020b) showed that completely random re-initialization of sparse subnetworks is also not viable. As the LTH initialization and mask are entangled, fully re-initializing will have us "lose the drawn lottery", resulting in the retrained sparse subnetwork converging to different (usually much poorer) solutions.

To balance between these concerns, we keep the original initialization intact for each parameter block and just re-scaled it by a learned scalar coefficient following recently proposed in (Zhu et al., 2021a). Aware of the sensitivity and negative impact of changing initialization identified by (Evci et al., 2020a), we point that that linear scaling will **not** hurt the original LTH initialization, thanks to the BatchNorm layer which will effectively absorb any linear scaling of the weights. More specifically, we optimized a small set of scalar coefficients to make the first update step (e.g., using SGD) as effective as possible at lowering the training loss. After the scalar coefficients are learned, the original LTH initializations of subnetworks are re-scaled and the optimization proceeds as normal.

3.3 Analysis: Commodity of our "tweaks"

In this section, we try to understand the implications of our techniques during training of sparse subnetworks, through some common lens. We study the loss landscape contours of our proposed techniques during the early training phase and late training phase and compare them with vanilla-LTH. Our methods can be viewed as a form of *learned smoothening* which can be incorporated at an early or late stage of the training. Smoothening tools can be applied on the logits (naive label-smoothening, knowledge distillation), on the weight dynamics (stochastic weight averaging), or on regularizing end solution. Sparse subnetworks trained from scratch suffer from high activation sparsity due to gradient clipping in the negative range of ReLU and it can be mitigated by replacing ReLU with smooth activations (Appendix A1.1). Figure 2 represents the top eigenvalue of the Hessian of model trained with different activation functions. Top eigenvalue of Hessian can act as a proxy for smoothness (Yao et al., 2020), and it is evident that ReLU-based sparse networks have comparatively high eigenvalue than smooth activation based subnetworks.

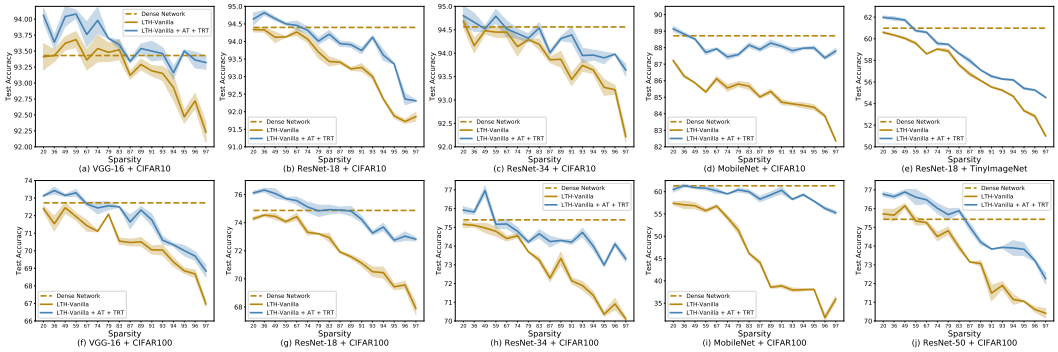


Figure 5: Comparison of testing accuracy of proposed techniques (AT + TRT combined) for Vgg16, ResNet-18, ResNet-34, ResNet-50, and MobileNet on CIFAR-10, CIFAR-100, Tiny-ImageNet datasets wrt. vanilla-LTH (three independent runs). Straight dash line represent the performance of dense network without any tweaking.

Other three of our proposed techniques smooth different angles of the sparse training. Figure 4 presents the comparison of the loss contours of ResNet-18 models (91% sparsity) in their very early training stage (epoch 5) on CIFAR-100. Figure 8 presents the loss landscape visualization of the trained models (epoch 180). It can be clearly observed that our architecture changes (skip-connection) can remarkably change the loss landscape (different counter shape, larger landscape area with different basin shape and size in middle) from beginning to end (Figure 4(b), 8(c)). As expected, our layer-wise scaling technique also has a very high impact in the early training stage (Figure 4(d)). We can observe a high difference in the variance of its contours compare to vanilla-LTH indicating the presence of smoothness. However, label-smoothening tends to impact the later phase of training more (highest difference in loss landscape, Figure 8(d)) compared to the early phase (minimal difference wrt. vanilla-LTH, Figure 4(c)). A more detailed analysis of top eigenvalues of the Hessian of models trained with our "tweaks" in available in (Appendix A1.7).

4 Experiments and Analysis

4.1 Settings

Datasets and Architecture: Following previous works of LTH (Frankle & Carbin, 2019; Frankle et al., 2019), we consider four popular architectures, ResNet-18, ResNet-34 (He et al., 2016), VGG-16 (Simonyan & Zisserman, 2014) and MobileNet (Howard et al., 2017) to evaluate the effectiveness of our proposed techniques. We considered three datasets in our experiments: CIFAR-10, CIFAR-100 (Krizhevsky et al., 2009), and Tiny-ImageNet and reported the performance of our techniques. In all our experiments, we randomly split the original training dataset into one training and one validation sets with a 9:1 ratio. Primary results, ablation, and visualizations are mainly performed on CIFAR-100 using ResNet-18.

Training and Evaluation Details: For all experiments, we by default use ResNet-18 and CIFAR-100 except except our extensive evaluation across datasets and architectures in Figure 5. For training both the dense and sparse network, we adopt an SGD optimizer with momentum 0.9 and weight decay $2e{-}4$. The initial learning rate is set to 0.1, and the networks are trained for 180 epochs with a batch size of 128. The learning rate decays by a factor of 10 at the 90th and 135th epoch during the training. The same training configurations are applied across all the other datasets and models evaluation in Figure 5. For weight rewinding experiments, we rewind the dense network weight to the 33rd epoch to initialize the sparse subnetwork.

4.2 Architecture Tweaking (AT) in LTH

We modify the sparse subnetwork found during the pruning process by introducing additional non-existing residual connections and changing the ReLU neurons with smoother activation functions such as Swish/Mish. Figure 1(a), (b) show the performance improvement of our AT techniques in

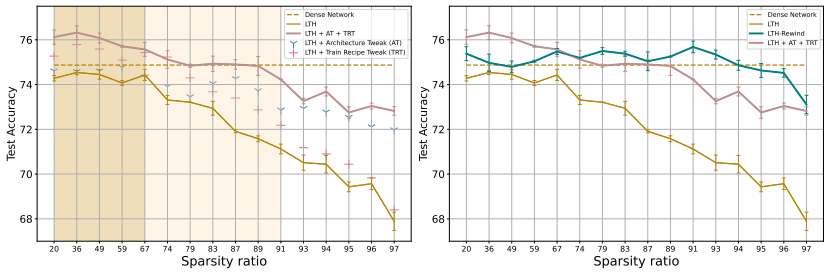


Figure 6: (a) Comparison of our techniques (architecture tweaking and training recipe tweaking) with respect to vanilla LTH using ResNet-18 on CIFAR-100 (three independent runs). We observe significant improvements in the performance of sparse networks when we combine our techniques during the *spending* process of tickets. "Winning tickets" can be identified for sparsity as high as 91% for ResNet-18 on CIFAR-100. (b) Comparison of test accuracy of LTH trained with our techniques and LTH-Rewind.

comparison of vanilla-LTH (Frankle & Carbin, 2019). Interestingly, the additional skip-connections hurts the performance of the dense network by -0.77%, but it significantly improves the performance of sparse networks, especially at extremely high sparsity levels, such as: $91\%(+1.05\%)$, $95\%(+2.58\%)$, and $97\%(+2.22\%)$. Similarly, the impact of smooth activation functions is marginal for dense networks: swish $(+0.21\%)$ and mish $(+0.24\%)$, but they also provide significant improvements in sparse network training. Figure 6(a) shows the performance when both AT techniques (skip connection + swish activation) are jointly applied in the sparse re-training step of LTH. We can observe that our techniques outperform vanilla-LTH markedly and the performance gap increase with the increase in sparsity level ($+4.08\%$ for sparsity level 97%).

4.3 Training Recipe Tweaking (TRT) in LTH

Sparse subnetwork training in LTH conventionally derives its training properties from its dense counterpart. We modify the training recipe of sparse subnetworks by changing their initialization (Zhu et al., 2021b) and using soft labels instead of one-hot labels. Due to minor performance differences between knowledge-distilled soft labels and naive label smoothening for sparse subnetworks, and the high intertwine of KD-labels with dense training, we choose to conduct our experiments with naive soft labels.

Figure 1 (c), (d) show the performance improvement of our training recipe tweaking techniques in comparison of vanilla-LTH (Frankle & Carbin, 2019). We can clearly observe that our techniques benefit sparse subnetworks significantly more than the dense network (dash straight line). Furthermore, they help "winning tickets" identified by LTH to win better, i.e have performance closer to the dense network. For example, winning tickets at 36% and 59% sparsity perform $+1.37\%$ and $+1.54\%$ better than vanilla-LTH due to label smoothening. Similarly, winning tickets at 59% and 74% sparsity perform $+0.53\%$ and $+0.89\%$ better than vanilla-LTH due to layer-wise scaled initialization. Figure 6(a) show the performance when our methods (scaled initialization + soft labels) are jointly applied in the sparse re-training step of LTH. It is evident from the figure that in the range of winning tickets, our methods significantly boost the performance of sparse subnetworks allowing them to perform even better than the dense network.

4.4 Combining AT and TRT

We combine our architecture (AT) and training recipe (TRT) tweaking techniques to train sparse subnetworks. Figure 6(a) and (Appendix A1.5) shows the performance of our combined approach wrt. to individual and vanilla-LTH. Very interestingly, it can be observed that while AT techniques provide significant improvements at high sparsity levels, TRT techniques provide significant improvements at low sparsity level. Due to the orthogonal benefits provided by AT and TRT methods, when they are combined, they patently outperform vanilla-LTH. We observe a huge performance improvement of $+4.93\%$ at sparsity level 97%. AT + TRT technique allow tickets identified at sparsity as high as 91% to be "winning tickets", in comparison to 67% for the vanilla-LTH.

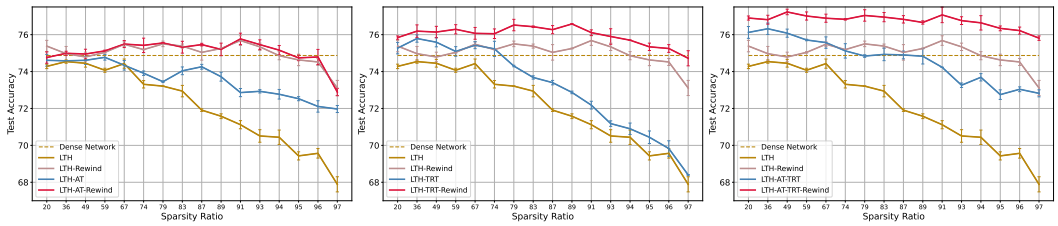


Figure 7: Comparison of the testing accuracy of our proposed techniques: *architecture tweaking (AT)* and *training recipe tweaking (TRT)* with respect to *rewinding* using ResNet-18 on CIFAR-100 (three independent runs). We also show the effect of combining rewinding with AT and TRT. State-of-the-art results are obtained for sparse retraining when our approaches are combined with rewinding.

Figure 5 demonstrate the performance of our proposal (AT + TRT) across ResNet-18, ResNet-34, ResNet-50 (He et al., 2016), VGG-16 (Simonyan & Zisserman, 2014) and MobileNet (Howard et al., 2017) on CIFAR-10, CIFAR-100, and Tiny-ImageNet. We compare our proposal with vanilla-LTH and found that our proposal out-performance prevails across all model architecture and datasets. Due to its simplicity and generalization capability across architectures and datasets, our proposal makes a strong case to be widely accepted for sparse subnetwork training.

Comparison with Weight rewinding: Weight rewinding (Frankle et al., 2019) proposes to "roll back" the dense model weights to some early training iteration, and use the rewound weight to initialize sparse subnetworks during the re-training step. While this technique has proven to be very helpful for stabilizing sparse re-training and attaining state-of-the-art performance for sparse subnetworks, it has its own drawbacks of the struggle to find rewind point, bookkeeping the weights of dense network during their training, entwine of rewind point to model architecture, etc. Figure 6(b) illustrates the performance comparison of AT + TRT (rose brown) with respect to weight rewinding technique (green). In order to compare the performance difference between them, we name the sparsity level where both lines cross each other as *critical sparsity*. At sparsity lower than it, AT + TRT generally outperforms weight rewinding. Furthermore, at some higher sparsity (74% - 89%, and 97%), AT + TRT can still perform comparably with rewinding.

4.5 Combining AT and TRT with Weight Rewinding: New State-of-the-art

In this section, we study the combination of our techniques: AT and TRT with weight rewinding. Our AT techniques provide significant improvement at high sparsity, while our TRT techniques help subnetworks with low sparsity. Being orthogonal, when they are combined, their performance is very comparable to weight rewinding. To further investigate how rewinding can improve the performance of our proposal, we perform experiments to combine both our proposal and weight rewinding technique together. In detail, we rewind the remaining weights of our pruned subnetwork to a specific early training point (18% in our experiments).

Figures 7 presents the results of our proposed techniques when they are combined with weight rewinding. We show the performance improvement with rewinding individually for AT and TRT, as well as when they are combined together. We observe that AT + rewinding performs marginally better than rewinding, but TRT + rewinding performs significantly better than rewinding. However, when AT + TRT together is combined with weight rewinding, we achieve state-of-the-art performance. Very interestingly, we observe a very marginal impact of sparsity on the performance. The performance of subnetworks does not vary much with increase in compression level. Overall performance is around $> 2.0\%$ than the dense network for almost all sparsity levels. The success of our proposed techniques with minimal changes in the current LTH paradigm, sets the new benchmark for sparse re-training and validates that sparse re-training needs not to intertwine with dense training; carefully re-tweaking the sparse re-training can be its important booster.

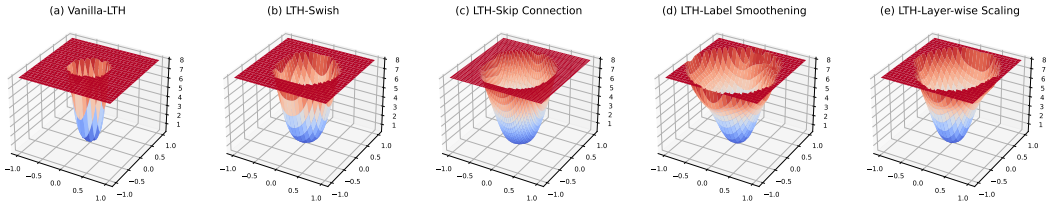


Figure 8: Comparison of loss landscape of models with 91% sparsity trained using vanilla-LTH and our Smoothness-aware techniques *(architecture tweaking and training recipe tweaking)*. Loss plots are generated with the same original images randomly chosen from CIFAR-100 test dataset using (Li et al., 2017a). z-axis denote the loss value which has been clamped at 8.0 for better visualization.

4.6 Ablation Study and Visualization

Naive LS verses learned logit smoothening As Knowledge Distillation (KD) can be viewed as a learned version of label smoothening (LS) (Yuan et al., 2020a), we tried to quantify whether there is any benefit of using KD (Hinton et al., 2015b) compared to naive LS (Szegedy et al., 2016) in our proposed LTH (+AT and +TRT). Figure 9 illustrates the benefits of replacing naive LS with KD (dense network serve as teacher). We found that KD has marginal benefits when it replaces naive LS in TRT of our proposed LTH. Also, the usage of KD leads to strong entwine between dense and sparse training, as well as an increase in computational overhead. Hence, we have conducted our experiments using naive LS by default.

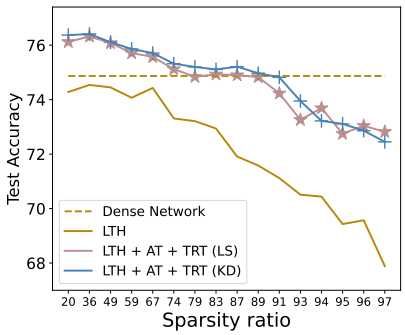


Figure 9: Comparison of our proposed LTH (+AT and +TRT), when naive soft labels (LS) are replaced with knowledge-distilled (KD) soft-labels in TRT for ResNet-18 on CIFAR-100.

Comparison of loss landscapes We expect our proposed techniques to find flatter minima for sparse subnetworks training to improve its generalization, and we show it happen by visualizing the loss landscape w.r.t. both input and the weight space. Figure 8 shows that our methods notably flatten the landscape w.r.t. input space, compared to vanilla-LTH baseline. Figure 10 follows (Yao et al., 2020) to perturb the trained model (91% sparsity) in weight space in direction of the top eigenvector and show how testing loss changes with perturbation distance. Our methods present better around the achieved local minima, which suggests improved generalization.

5 Conclusion

This paper makes a contrary argument to the common wisdom in LTH that sparse re-training has to stick to the protocol of dense network training. We demonstrated that purposely re-tweaking the network architecture, initialization, and training protocol from dense training can actually improve spare subnetwork performance. We present two groups of techniques - architecture tweaking and training recipe tweaking, both motivated by injecting smootheness in training sparse subnetworks. Our extensive experiments across several datasets and architectures present the generalization capability of our techniques to achieve SOTA performance. Our future work will aim for more theoretical understanding of the role of our techniques in sparse (re-)training performance improvement.

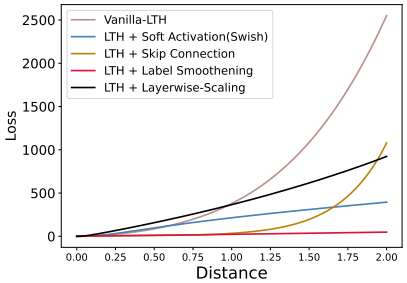


Figure 10: The change in testing loss as a function of perturbed weight distance, in the direction of top eigenvector of Hessian matrix for ResNet-18 trained on CIFAR-100.

REFERENCES

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Michael Carbin, and Zhangyang Wang. The lottery tickets hypothesis for supervised and self-supervised pre-training in computer vision models. *arXiv preprint arXiv:2012.06908*, 2020a.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained bert networks. *arXiv preprint arXiv:2007.12223*, 2020b.

Tianlong Chen, Yu Cheng, Zhe Gan, Jingjing Liu, and Zhangyang Wang. Ultra-data-efficient gan training: Drawing a lottery ticket first, then training it toughly. *arXiv preprint arXiv:2103.00397*, 2021a.

Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. A unified lottery ticket hypothesis for graph neural networks, 2021b.

Tianlong Chen, Zhenyu Zhang, Sijia Liu, Shiyu Chang, and Zhangyang Wang. Long live the lottery: The existence of winning tickets in lifelong learning. In *International Conference on Learning Representations*, 2021c. URL https://openreview.net/forum?id=LXMSvPmsm0g.

Xiaohan Chen, Yu Cheng, Shuohang Wang, Zhe Gan, Zhangyang Wang, and Jingjing Liu. Earlybert: Efficient bert training via early-bird lottery tickets. *arXiv preprint arXiv:2101.00063*, 2020c.

Xuxi Chen, Zhenyu Zhang, Yongduo Sui, and Tianlong Chen. Gans can play lottery tickets too. In *International Conference on Learning Representations*, 2021d. URL https://openreview.net/forum?id=1AoMhc_9jER.

Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13733–13742, 2021.

Utku Evci, Yani A Ioannou, Cem Keskin, and Yann Dauphin. Gradient flow in sparse neural networks and how lottery tickets win. *arXiv preprint arXiv:2010.03533*, 2020a.

Utku Evci, Yani Andrew Ioannou, Cem Keskin, and Yann Dauphin. Gradient flow in sparse neural networks and how lottery tickets win. *ArXiv*, abs/2010.03533, 2020b.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*, 2019.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015b.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 41–50, 2019.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015a.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531, 2015b.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

Jack SN Jean and Jin Wang. Weight smoothing to improve network generalization. *IEEE Transactions on neural networks*, 5(5):752–763, 1994.

Neha Mukund Kalibhat, Yogesh Balaji, and Soheil Feizi. Winning lottery tickets in deep generative models. *arXiv preprint arXiv:2010.02350*, 2020.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. Visualizing the loss landscape of neural nets. *CoRR*, abs/1712.09913, 2017a. URL `http://arxiv.org/abs/1712.09913`.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*, 2017b.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019.

Haoyu Ma, Tianlong Chen, Ting-Kuei Hu, Chenyu You, Xiaohui Xie, and Zhangyang Wang. Good students play big lottery better. *arXiv preprint arXiv:2101.03255*, 2021.

Diganta Misra. Mish: A self regularized non-monotonic neural activation function. *arXiv preprint arXiv:1908.08681*, 4:2, 2019.

Rafael Müller, Simon Kornblith, and Geoffrey Hinton. When does label smoothing help? *arXiv preprint arXiv:1906.02629*, 2019.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

Sai Prasanna, Anna Rogers, and Anna Rumshisky. When bert plays the lottery, all tickets are winning. *arXiv preprint arXiv:2005.00561*, 2020.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *International Conference on Learning Representations*, 2020.

Pedro Savarese, Hugo Silva, and Michael Maire. Winning the lottery with continuous sparsification. In *NeurIPS*, 2020.

Zhiqiang Shen, Zechun Liu, Dejia Xu, Zitian Chen, Kwang-Ting Cheng, and Marios Savvides. Is label smoothing truly incompatible with knowledge distillation: An empirical study. *arXiv preprint arXiv:2104.00676*, 2021.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Hidenori Tanaka, Daniel Kunin, Daniel LK Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *arXiv preprint arXiv:2006.05467*, 2020.

Kale-ab Tessera, Sara Hooker, and Benjamin Rosman. Keep the gradients flowing: Using gradient flow to study sparse network optimization. *arXiv preprint arXiv:2102.01670*, 2021.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *ArXiv*, abs/1706.03762, 2017.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.

Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W. Mahoney. Pyhessian: Neural networks through the lens of the hessian. *2020 IEEE International Conference on Big Data (Big Data)*, pp. 581–590, 2020.

Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Richard G. Baraniuk, Zhangyang Wang, and Yingyan Lin. Drawing early-bird tickets: Toward more efficient training of deep networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJxsrgStvr.

Haonan Yu, Sergey Edunov, Yuandong Tian, and Ari S Morcos. Playing the lottery with rewards and multiple languages: lottery tickets in rl and nlp. *arXiv preprint arXiv:1906.02768*, 2019.

Li Yuan, Francis Tay, Guilin Li, Tao Wang, and Jiashi Feng. Revisiting knowledge distillation via label smoothing regularization. pp. 3902–3910, 06 2020a. doi: 10.1109/CVPR42600.2020.00396.

Li Yuan, Francis EH Tay, Guilin Li, Tao Wang, and Jiashi Feng. Revisiting knowledge distillation via label smoothing regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3903–3911, 2020b.

Chen Zhu, Renkun Ni, Zheng Xu, Kezhi Kong, W. R. Huang, and T. Goldstein. Gradinit: Learning to initialize neural networks for stable and efficient training. *ArXiv*, abs/2102.08098, 2021a.

Chen Zhu, Renkun Ni, Zheng Xu, Kezhi Kong, W. Ronny Huang, and Tom Goldstein. Gradinit: Learning to initialize neural networks for stable and efficient training. *CoRR*, abs/2102.08098, 2021b. URL https://arxiv.org/abs/2102.08098.

A1 MORE EXPERIMENT RESULTS

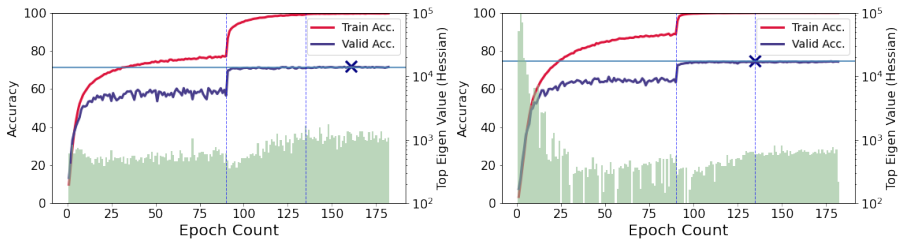
A1.1 ANALYSIS OF ACTIVATION SPARSITY OF RELU AND SMOOTH NEURONS

Sparse neural networks suffer from very high activation sparsity due to the non-smooth nature of ReLU. ReLU gradient changes suddenly around zero, and this causes high proportions of network activations to be clamped to zero. Table A1 provide the activation sparsity of ReLU at different sparsity level for ResNet-18 model. Activation sparsity of the network increases with the depth for ReLU activation. When ReLU is replaced with smooth activation (Swish), the activation sparsity of the network drops below 1%. Soft-activations allows sparse NN training to get rid of zero effect of ReLU (zero activations) and facilitate smoother and healthy gradient flow in the network which may help the performance of sparse NN training.

Table A1: Activation sparsity of different layers of ResNet-18 trained on CIFAR-100 using ReLU and Swish at various sparsity levels.

Sparsity Level	Layer 1		Layer 2		Layer 3		Layer 4	
	ReLU	Swish	ReLU	Swish	ReLU	Swish	ReLU	Swish
20%	28.93%	0.33%	49.23%	0.25%	62.36%	0.22%	65.50%	0.16%
36%	32.29%	0.31%	47.46%	0.25%	60.40%	0.23%	65.69%	0.16%
67%	35.18%	0.29%	46.29%	0.25%	58.01%	0.23%	64.19%	0.16%
74%	35.70%	0.31%	50.50%	0.25%	56.30%	0.23%	64.47%	0.16%
96%	27.43%	0.30%	38.26%	0.26%	39.78%	0.27%	56.06%	0.23%
97%	27.99%	0.39%	36.63%	0.24%	39.59%	0.28%	53.18%	0.24%

A1.2 HESSIAN VISUALIZATION DURING SPARSE NN RETRAINING



(a) ResNet-18 + CIFAR100 [Vanilla-LTH] (b) ResNet-18 + CIFAR100 [Proposed-LTH]

Figure A11: Comparison of top eigenvalue of Hessian (mean across 3 batches every epoch) during training of our proposed LTH (+AT and +TRT) and vanilla-LTH (91% sparsity). Left y-axis denote accuracy and Right y-axis denote top eigen value of Hessian in log scale. Dotted vertical line denote the epoch when learning rate is scaled by 0.1 and solid horizontal line denote test accuracy. Cross-sign indicate the epoch when best validation accuracy is achieved. The eigenvalues of hessian is calculated using (Yao et al., 2020).

Top eigenvalues of hessian can be used to analyze the toplogy of the loss landscape (i.e., curvature information) and understand the behaviour of different models/optimizers. Figure A11 presents the top eigenvalues of hessian our proposed LTH and vanilla-LTH (91% sparsity) for each training epoch. We can observe that in initial phase of sparse subnetwork training, our methods provide high push (high eigenvalue) and soon land up in a smoother regime (low eigenvalue). Due to the initial push, our method possibly achieve higher train/val accuracy in initial phase of training (before learning rate scaling) in compare to vanilla-LTH, and ultimately a significant improvement of $+3.11\%$ in test accuracy. Note that during final stage of training, our proposed LTH have high low eigenvalue than vanilla-LTH, which indicate the presence of high smoothness.

A1.3 DETAILED RESULTS FOR OUR "TWEAKS" ON VANILLA-LTH:

In this section, we present the detailed improvements of our architecture tweaks and training recipe tweaks. The results are for CIFAR-100 dataset trained with ResNet-18 architecture. Note that we have used exactly same set of hyperparameters to train our tweaked networks which have been used to train vanilla-LTH for fair comparison.

Table A2: Results of testing accuracy for ResNet-18 on CIFAR-100 for our architecture tweaking and training recipe tweaking techniques. Results are mean across 3 runs with different seeds (42, 1099, 5469). All the networks are trained with exactly same settings (Section 4.1) for 180 epoch. Note that 0% sparsity row indicate the performance of the dense network when trained with/without our tweaks .

Sparsity	LTH	LTH + Skip	LTH + Swish	LTH + Scaling	LTH + LS
0%	74.87%	74.10% (-0.77)	75.08% (+0.21)	74.82% (-0.05)	75.42% (+0.55)
20%	74.28%	73.99% (-0.29)	74.95% (+0.67)	74.62% (+0.34)	75.90% (+1.62)
36%	74.54%	75.03% (+0.49)	75.35% (+0.81)	74.74% (+0.20)	75.91% (+1.37)
49%	74.45%	74.56% (+0.11)	74.34% (-0.11)	74.50% (+0.05)	75.54% (+1.09)
59%	74.07%	73.91% (-0.16)	74.94% (+0.87)	74.60% (+0.53)	75.61% (+1.54)
67%	74.43%	74.08% (-0.35)	74.62% (+0.19)	74.63% (+0.20)	75.46% (+1.03)
74%	73.31%	73.94% (+0.63)	73.37% (+0.06)	74.20% (+0.89)	74.69% (+1.38)
79%	73.21%	73.77% (+0.56)	73.45% (+0.24)	73.93% (+0.72)	74.58% (+1.37)
83%	72.94%	73.69% (+0.75)	73.22% (+0.28)	73.12% (+0.18)	74.12% (+1.18)
87%	71.91%	72.86% (+0.95)	73.27% (+1.36)	72.10% (+0.19)	72.65% (+0.74)
89%	71.58%	72.42% (+0.84)	72.67% (+1.09)	72.09% (+0.51)	72.26% (+0.68)
91%	71.12%	72.17% (+1.05)	72.43% (+1.31)	71.83% (+0.71)	71.13% (+0.01)
93%	70.51%	72.15% (+1.64)	71.98% (+1.47)	71.35% (+0.84)	71.24% (+0.73)
94%	70.44%	72.17% (+1.73)	71.33% (+0.89)	70.65% (+0.21)	70.47% (+0.03)
95%	69.43%	72.01% (+2.58)	70.61% (+1.18)	70.10% (+0.67)	69.61% (+0.18)
96%	69.57%	71.71% (+2.14)	69.87% (+0.30)	69.62% (+0.05)	69.51% (-0.06)
97%	67.89%	70.11% (+2.22)	69.01% (+1.12)	68.45% (+0.56)	68.58% (+0.69)

A1.4 GENERALIZATION ACROSS OMP

We investigated the effectiveness of our proposed techniques when *iterative magnitude pruning* is replaced with *one-shot magnitute pruning* (OMP). Figure A12 illustrate the generalization of the benefits of our proposed techniques when IMP is replaced with OMP. We observe that our techniques still perform significantly better than vanilla-LTH when IMP is replaced with OMP. We would like to highlight that our methods are independent of mask finding and can be easily plugged with subnetworks identified by any pruning algorithm for performance gain.

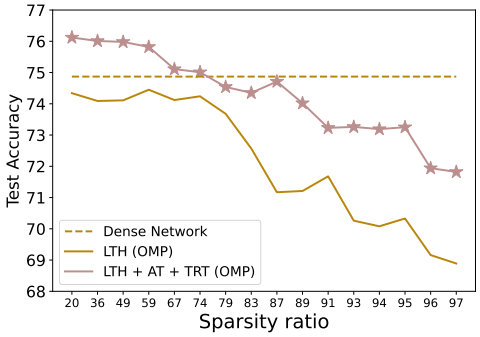


Figure A12: Testing accuracy of ResNet-18 with our proposed LTH (+AT and +TRT) when IMP is replaced with OMP.

A1.5 DETAILED RESULTS FOR COMPOSITION OF OUR "TWEAKS" AND COMPARISON WITH WEIGHT REWINDING

In this section, we present the exact numerical values of the performance of our tweaks (architecture tweaks, training recipe tweaks) and compared it with weight rewinding.

Table A3: Results of testing accuracy for ResNet-18 on CIFAR-100 for the composition of architecture tweaking (AT) and training recipe tweaking (TRT) techniques. All the networks are trained with exactly same settings (Section 4.1) for 180 epoch. The last column indicates the performance difference of our method with respect to weight rewinding.

Sparsity	LTH	LTH + AT	LTH + TRT	LTH + AT + TRT	LTH Rewind	Difference
20%	74.28%	74.61%	75.27%	76.12%	75.38%	+0.74%
36%	74.54%	74.58%	75.79%	76.32%	74.97%	+1.35%
49%	74.45%	74.61%	75.59%	76.08%	74.79%	+1.29%
59%	74.07%	74.77%	75.09%	75.71%	75.04%	+0.67%
67%	74.43%	74.34%	75.43%	75.57%	75.48%	+0.09%
74%	73.31%	73.91%	75.23%	75.12%	75.19%	-0.07%
79%	73.21%	73.45%	74.3%	74.84%	75.5%	-0.66%
83%	72.94%	74.04%	73.68%	74.93%	75.38%	-0.45%
87%	71.91%	74.26%	73.4%	74.9%	75.04%	-0.14%
89%	71.58%	73.72%	72.87%	74.83%	75.25%	-0.42%
91%	71.12%	72.86%	72.18%	74.23%	75.68%	-1.45%
93%	70.51%	72.93%	71.18%	73.26%	75.34%	-2.08%
94%	70.44%	72.77%	70.9%	73.69%	74.86%	-1.17%
95%	69.43%	72.53%	70.44%	72.75%	74.63%	-1.88%
96%	69.57%	72.11%	69.83%	73.04%	74.52%	-1.48%
97%	67.89%	71.97%	68.4%	72.82%	73.11%	-0.29%

A1.6 Comparison of our "Tweaks" incorporated LTH with respect to other pruning algorithm

We compare our tweaks incorporated LTH methods with other popular pruning algorithms using ResNet-32 on CIFAR-10/100. We have used the official implementation of compared algorithms and trained them with default best setting suggested by their respective papers.

Table A4: Comparison of results of testing accuracy for ResNet-34 on CIFAR-10/100 for various pruning algorithms with our "Tweaks" incorporated LTH.

	CIFAR-10			CIFAR-100		
Sparsity Level	90%	95%	98%	90%	95%	98%
Resnet-32 [No Pruning]		94.80%			74.64%	
Random Pruning	89.95%	89.68%	86.13%	63.13%	64.55%	19.83%
GraSP	92.20%	91.39%	88.70%	69.24%	66.50%	58.43%
SynFlow	92.01%	91.67%	88.10%	69.03%	65.23%	58.73%
LTH + AT + TRT	**94.31%**	**93.90%**	**93.64%**	**74.30%**	**73.99%**	**72.22%**

A1.7 HESSIAN ANALYSIS OF OUR PROPOSED "TWEAKS"

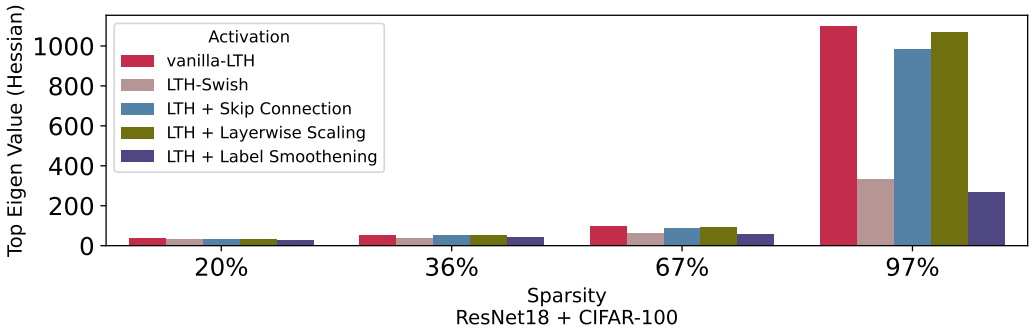


Figure A13: Comparison of the top eigenvalue of Hessian (mean across 10 random batches) of sparse NN trained with our proposed "tweaks". Comparatively smaller eigenvalues of the models trained with incorporating our "tweaks" wrt. vanilla-LTH the presence of high smoothness. The eigenvalues of hessian are calculated using (Yao et al., 2020).