RETHINKING AGAIN THE VALUE OF NETWORK PRUNING - A DYNAMICAL ISOMETRY PERSPECTIVE

Anonymous authors
Paper under double-blind review

ABSTRACT

Several recent works questioned the value of inheriting weights in structured neural network pruning because they empirically found training from scratch can match or even outperform finetuning a pruned model. In this paper, we present evidence that this argument is actually *inaccurate* because of using improperly small finetuning learning rates. With larger learning rates, our results consistently suggest pruning outperforms training from scratch on multiple networks (ResNets, VGG11) and datasets (MNIST, CIFAR10, ImageNet) over most pruning ratios. To deeply understand why finetuning learning rate holds such a critical role, we examine the theoretical reason behind through the lens of *dynamical isometry*, a nice property of networks that can make the gradient signals preserve norm during propagation. Our results suggest that weight removal in pruning breaks dynamical isometry, *which fundamentally answers for the performance gap between a large finetuning LR and a small one*. Therefore, it is necessary to recover the dynamical isometry before finetuning. In this regard, we also present a regularization-based technique to do so, which is rather simple-to-implement yet effective in dynamical isometry recovery on modern residual convolutional neural networks.

1 INTRODUCTION

Pruning is a time-honored methodology to reduce parameters in a neural network without seriously compromising its performance (Reed, 1993; Sze et al., 2017). The prevailing pipeline of pruning comprises three steps: 1) **pretraining**: train a dense model; 2) **pruning**: prune the dense model based on certain rules; 3) **finetuning**: retrain the pruned model to regain performance. Most existing research focuses on the second step, seeking the best criterion to select unimportant weights so as to incur as less performance degradation as possible. This 3-step pipeline has been practiced for more than 30 years (Mozer & Smolensky, 1989; LeCun et al., 1990) and is still extensively adopted in today's pruning methods (Sze et al., 2017).

These said, several recent works (Crowley et al., 2018; Liu et al., 2019) questioned the necessity of inheriting weights from a pretrained model because they empirically found the small model trained from scratch can match (or sometimes outperform) the counterpart pruned from the pre-trained large model. This acutely challenges the past wisdom as well as our common belief about pruning. As far as we know, there is *no* formal response to this critical conflict. A theoretical-level understanding of this problem is even more elusive.

Meanwhile, the pruning community has been observing even more open questions. Specifically, (Renda et al., 2020; Le & Hua, 2021) found that the learning rate (LR) in finetuning holds a critical role in the final performance. A proper learning rate schedule (*e.g.*, a larger initial LR 10^{-2} vs. 10^{-3} with step-decay schedule) can improve the top-1 accuracy of a pruned ResNet-34 model (He et al., 2016) by more than 1% on ImageNet (Deng et al., 2009). This discovery calls for more attention being paid to the finetuning step when comparing different pruning methods. Unfortunately, they did not present more theoretical insights to explain its occurrence. This also remains an open question in the community up to date.

In this paper, we will show these two open questions actually point to the same one. Specifically, we rerun the experiments of (Crowley et al., 2018; Liu et al., 2019) and find simply using a larger finetuning LR (10^{-2} vs. 10^{-3} and decay it) can *significantly* improve the final performance. Compared to the improved pruning performance, training from scratch does *not* compete or surpass pruning

anymore (see Tab. 1 and Tab. 2 on ImageNet). This observation invites many questions immediately: (1) Theoretical understanding: Why does this happen? What is the theoretical reason behind it? (2) Practical solution: If this is a problem, how to fix it? Can the understanding of this problem lead us to better pruning algorithms?

This paper will present answers to all these questions. The key tool we employ to unveil the mysteries is *dynamical isometry* (Saxe et al., 2014), which describes a kind of nice property in neural networks that are easy to optimize. We carefully design an explanatory experiment using a linear MLP (multi-layer perceptron) network to demonstrate how finetuning LR affects the final performance by affecting dynamical isometry. In brief, we observe the finetuning process can recover dynamical isometry; a larger LR can help recover it faster (or better), hence the better final performance. The proposed explanation is validated by our empirical results and resonates with many empirical observations. Furthermore, by the explanation, we learn dynamical isometry recovery is rather imperative. To achieve so, we present a very simple regularization-based method for pruning and show its effectiveness in recovering dynamical isometry on modern residual convolutional neural networks (CNNs).

Contributions. (1) We empirically demonstrate the questioning about the value of inheriting weights in structured pruning in previous works is inaccurate and point out that the direct cause is improperly using a small finetuning LR. Our finding justifies the value of inheriting weights in structured pruning. (2) On top of the empirical finding, more importantly, we present a theoretical explanation through examining the dynamical isometry of networks in pruning. This explanation is empirically validated by our carefully designed control experiments. (3) In addition to the theoretical understanding, we also propose a regularization-based method for dynamical isometry recovery. Despite its brutal simplicity, it is shown effective to recover the broken dynamical isometry on modern residual convolutional neural networks.

2 Related Work

Conventional pruning. Pruning aims to remove as many parameters as possible in a neural network meanwhile maintaining its performance. There are many ways to categorize pruning methods. The most popular two are grouping by pruning structure and methodology.

(1) In terms of pruning structure, pruning can be specified into unstructured pruning (Han et al., 2015; 2016) and structured pruning (Wen et al., 2016; Li et al., 2017; He et al., 2017). For the former, a single weight is the basic pruning element. Unstructured pruning can deliver a high compression ratio; whereas, without regularization, the pruned locations usually spread *randomly* in the network, which is hard to exploit for acceleration. On the opposite, structured pruning introduces certain patterns in the pruned locations, which benefit subsequent acceleration while cannot achieve as much compression. Choices between unstructured and structured pruning depend on specific application needs. For structured pruning, there are still many sub-groups (Mao et al., 2017). In the literature, without specific mention, structured pruning means filter pruning or channel pruning. This paper focuses on **structured (filter) pruning** because the "no value of inheriting weights" argument is mainly discussed in this context (Liu et al., 2019).

(2) In terms of pruning methodology (*i.e.*, how to select unimportant weights to prune), pruning falls into two paradigms in general: importance-based and penalty-based. The former prunes weights based on some established importance criteria, such as magnitude (for unstructured pruning) (Han et al., 2015; 2016) or L_1-norm (for filter pruning) (Li et al., 2017), saliency based on 2nd-order gradients (*e.g.*, Hessian or Fisher) (LeCun et al., 1990; Hassibi & Stork, 1993; Theis et al., 2018; Wang et al., 2019a; Singh & Alistarh, 2020). The latter adds a penalty term to the objective function, drives unimportant weights towards zero, then removes those with the smallest magnitude. Note, the two groups are *not* starkly separated. Many methods take wisdom from both sides. For example, (Ding et al., 2018; Wang et al., 2019b; 2021b) select unimportant weights by magnitude (akin to the first group) while also employing the regularization to penalize weights (akin to the second group). There is no conclusion about which paradigm is better, yet empirically, the state-of-the-art pruning methods are closer to the second paradigm, *i.e.*, deciding weights via training instead of some derived formulas. Although no theories have formally discussed the reason, we can take a rough guess with the knowledge from this paper: Training can recover dynamical isometry (which is our finding in Fig. 2), which is beneficial to subsequent finetuning.

For more comprehensive literature, we refer interested readers to several surveys: an outdated one (Reed, 1993), some recent surveys of pruning alone (Gale et al., 2019; Blalock et al., 2020) or pruning as a sub-topic under the general umbrella of model compression and acceleration (Sze et al., 2017; Cheng et al., 2018a;b; Deng et al., 2020).

Pruning at initialization (PaI). Recent years have seen several new pruning paradigms. The most prominent one is pruning at initialization. Different from the conventional pruning, which prunes a *pretrained* model, PaI methods prune a *randomly initialized* model. Existing PaI approaches mainly include (Lee et al., 2019; 2020; Wang et al., 2020; Frankle et al., 2021; Ramanujan et al., 2020) and the series of lottery ticket hypothesis (Frankle & Carbin, 2019; Frankle et al., 2020). Interested readers may refer to (Wang et al., 2021a) for a comprehensive summary about PaI.

This topic is relevant to this work mainly because one PaI paper (Lee et al., 2020) also examines pruning using the tool of dynamical isometry. The similarity between our paper and theirs is that we both employ dynamical isometry as a tool to examine the property of network pruning. However, our paper is *significantly different* from theirs in many axes: (1) Basic setting. The most obvious difference is that we focus on pruning a *pretrained* model while (Lee et al., 2020) focuses on pruning at initialization (PaI). They are two different tracks in pruning (as such, PaI methods typically do *not* compare with the methods of pruning pretrained models) and the latter was shown to consistently underperform the former (Frankle et al., 2021; Wang et al., 2021a). (2) Motivation. Despite the same tool (mean JSV), (Lee et al., 2020) uses it to select unimportant weights to prune (*i.e.*, for a new pruning criterion), while we use it to analyze why finetuning LR has a significant impact on final performance. The role of finetuning LR in pruning is *not* mentioned at all in their paper. (3) Proposed technical method. (Lee et al., 2020) focuses on *unstructured pruning*, while we focuses on *structured pruning*. This further leads to fundamental difference when designing the dynamical isometry recovery (DIR) methods – In (Lee et al., 2020), their proposed method is to use iterative optimization for *approximated* isometry (due to the irregular sparsity); while in our case, since the pruned filers can be completely removed from the network, one of our DIR method (OrthP) has *closed-form* solution and can achieve *exact* isometry. (4) Finally, in terms of empirical results, (Lee et al., 2020) only conducts experiments on MNIST (LeCun et al., 1998) and CIFAR (Krizhevsky, 2009), while we have extensive results on the large-scale ImageNet dataset (Deng et al., 2009).

2.1 Empirical Study: Larger finetuning LR is Critical

As far as we know, mainly *two* papers question the value of inheriting weights from a pretrained model: (Crowley et al., 2018; Liu et al., 2019). Both papers draw two similar conclusions. (1) Inheriting weights from a pretrained model in pruning has *no* value, *i.e.*, training from scratch the small model can match (or outperform sometimes) the counterpart pruned from a big pretrained model. (2) Given the fact of (1), what really matters in pruning may lie in the pruned *architecture* instead of the inherited weight values. As such, both papers propose to view pruning as a form of neural architecture search (Zoph & Le, 2017; Elsken et al., 2019). In this section, we first reexamine the empirical studies in (Crowley et al., 2018; Liu et al., 2019) to show that the "no value of inheriting weights" argument is actually inaccurate owing to improper finetuning LR schedules.

Reexamination of (Liu et al., 2019). Before presenting results, here are some important comparison setting changes worth particular attention: (1) In (Liu et al., 2019), they compare training from scratch with *six* pruning methods (five structured pruning methods (Li et al., 2017; Luo et al., 2017; Liu et al., 2017; He et al., 2017; Huang & Wang, 2018) and one unstructured pruning method (Han et al., 2015)). Here, we only focus on the L_1*-norm pruning* (Li et al., 2017) on ImageNet. The main reason is that, L_1-norm pruning is well-known a very *basic* filter pruning method. If we can show it outperforms training from scratch already, it will be no surprise to see other more advanced pruning methods also outperform training from scratch. In this sense, L_1-norm pruning is the most representative method here for our investigation. (2) In (Liu et al., 2019), they have two variants for the number of epochs in scratch training, "Scratch-E" and "Scratch-B". For the former, different small models are trained for a fixed number of epochs; for the latter, *smaller* models are trained for *more* epochs to maintain the same computation budget (Scratch-B was shown to be better than Scratch-E in (Liu et al., 2019)). Also, they decay LR only to 10^{-3} following the official PyTorch ImageNet example[1]. Here, we simply train all the networks for the same number of epochs but

[1] https://github.com/pytorch/examples/tree/master/imagenet

Table 1: Top-1 accuracy comparison of different implementations of the L_1-norm pruning (Li et al., 2017) on ImageNet. We adopt the torchvision models as unpruned models for fair comparison. ResNet-34-A speedup: $1.18\times$. ResNet-34-B speedup: $1.32\times$. The results of (Li et al., 2017) and (Liu et al., 2019) are directly cited from their papers. The best cases in our training from scratch and pruning are randomly repeated for 3 times ($\pm$ indicates stddev) to prevent random variation.

Implementation	Unpruned (%)	Pruned model	Scratch (%)	Pruned-Finetuned (%)	Finetuning LR schedule
(Li et al., 2017)	73.23	ResNet-34-A	(Not reported)	72.56	20 epochs, initial 10^{-3}, fixed
		ResNet-34-B	(Not reported)	72.17	20 epochs, initial 10^{-3}, fixed
(Liu et al., 2019)	73.31	ResNet-34-A	**73.03**	72.56	20 epochs, initial 10^{-3}, fixed
		ResNet-34-B	**72.91**	72.29	20 epochs, initial 10^{-3}, fixed
Our rerun	73.31	ResNet-34-A	$73.51_{\pm 0.12}$	72.91	20 epochs, initial 10^{-3}, fixed
				72.94	90 epochs, initial 10^{-3}, fixed
				73.88	90 epochs, initial 10^{-3}, decay
				$\mathbf{73.92_{\pm 0.03}}$	90 epochs, initial 10^{-2}, decay
Our rerun	73.31	ResNet-34-B	$73.16_{\pm 0.12}$	72.50	20 epochs, initial 10^{-3}, fixed
				72.58	90 epochs, initial 10^{-3}, fixed
				73.61	90 epochs, initial 10^{-3}, decay
				$\mathbf{73.62_{\pm 0.04}}$	90 epochs, initial 10^{-2}, decay

Table 2: Top-1 accuracy comparison between scratch training ("Scratch") and L_1-norm pruning (Li et al., 2017) on ImageNet. "PR" means pruning ratio. †We adopt the official torchvision models as unpruned models. "Finetuned-1" and "Finetuned-2" refers to two finetuning LR schedules ("Finetuned-1": 90 epochs, initial 10^{-2}, decay $30/60/75$; "Finetuned-2": 90 epochs, initial 10^{-3}, decay $45/68$). Best results are in **bold**, second best underlined, each averaged by 3 random runs.

Network	PR	Params reduc. (%)	FLOPs reduc. (%)	Scratch (%)	Pruned-Finetuned-1 (%)	Pruned-Fintuned-2 (%)
ResNet-18	0	0	0	69.76†	/	/
	0.1	9.56	9.58	$70.15_{\pm 0.02}$	$70.43_{\pm 0.02}$	$\mathbf{70.48_{\pm 0.10}}$
	0.3	28.32	28.18	$68.90_{\pm 0.10}$	$69.29_{\pm 0.07}$	$\mathbf{69.54_{\pm 0.09}}$
	0.5	47.03	46.20	$67.03_{\pm 0.01}$	$67.36_{\pm 0.03}$	$\mathbf{67.71_{\pm 0.05}}$
	0.7	65.99	64.93	$64.21_{\pm 0.10}$	$63.72_{\pm 0.03}$	$\mathbf{64.45_{\pm 0.04}}$
	0.9	84.75	83.52	$\mathbf{56.70_{\pm 0.17}}$	$53.49_{\pm 0.10}$	$55.89_{\pm 0.11}$
	0.95	89.51	88.03	$\mathbf{51.83_{\pm 0.14}}$	$44.46_{\pm 0.15}$	$49.99_{\pm 0.10}$
ResNet-34	0	0	0	73.31†	/	/
	0.1	9.84	9.92	$73.53_{\pm 0.10}$	$73.86_{\pm 0.05}$	$\mathbf{74.04_{\pm 0.05}}$
	0.3	29.15	29.26	$72.50_{\pm 0.17}$	$73.11_{\pm 0.06}$	$\mathbf{73.31_{\pm 0.08}}$
	0.5	48.41	48.12	$71.27_{\pm 0.03}$	$71.71_{\pm 0.06}$	$\mathbf{71.85_{\pm 0.07}}$
	0.7	67.95	67.63	$68.69_{\pm 0.10}$	$68.90_{\pm 0.08}$	$\mathbf{69.33_{\pm 0.04}}$
	0.9	87.26	86.97	$62.08_{\pm 0.12}$	$60.34_{\pm 0.03}$	$\mathbf{62.26_{\pm 0.06}}$
	0.95	92.16	91.69	$\mathbf{57.21_{\pm 0.15}}$	$52.83_{\pm 0.11}$	$56.69_{\pm 0.23}$
VGG11_BN	0	0	0	70.37†	/	/
	0.1	9.19	17.78	$68.51_{\pm 0.04}$	$71.45_{\pm 0.07}$	$\mathbf{71.74_{\pm 0.04}}$
	0.3	26.80	47.63	$66.60_{\pm 0.11}$	$70.00_{\pm 0.03}$	$\mathbf{70.53_{\pm 0.05}}$
	0.5	43.86	70.56	$65.85_{\pm 0.13}$	$67.34_{\pm 0.05}$	$\mathbf{68.01_{\pm 0.10}}$
	0.7	60.54	87.03	$61.56_{\pm 0.06}$	$62.14_{\pm 0.09}$	$\mathbf{63.14_{\pm 0.08}}$
	0.9	76.50	96.49	$48.34_{\pm 0.12}$	$45.11_{\pm 0.07}$	$\mathbf{48.52_{\pm 0.06}}$
	0.95	80.49	97.76	$35.47_{\pm 0.09}$	$33.50_{\pm 0.10}$	$\mathbf{38.47_{\pm 0.13}}$

ensure the epochs are abundant (120 epochs) and decay LR to a very small amount (10^{-5}). These two changes are to make sure the networks are trained to *full convergence*. As we will show, one primary cause possibly leading (Liu et al., 2019) to an inaccurate conclusion is exactly that the pruned networks are *not* fully converged (see Tab. 1).

With the LR schedule changes, we rerun the experiments *using the released code of (Liu et al., 2019)*. Results are presented in Tab. 1. In the implementations of (Liu et al., 2019), the finetuned model is outperformed by the scratch training one, hence their "no value of inheriting weights" argument. We also reproduce their settings (the two rows of "20 epochs, initial 10^{-3}, fixed" in "Our rerun") for confirming their argument. However, the finetuning LR schedule "20 epochs, initial 10^{-3}, fixed" is actually sub-optimal; the network is *not* fully converged. Using the proper ones ("90 epochs, initial 10^{-3}, decay" or "90 epochs, initial 10^{-2}, decay"), pruning *outperforms* training from

scratch for both ResNet-34-A and ResNet-34-B. (We note the pruned models even outperform the original models. This is probably because pruning reduces the network redundancy, thus curbing overfitting. This phenomenon is also widely observed in past pruning works (Han et al., 2016; Wen et al., 2016; He et al., 2017) especially under small pruning ratios as in Tab. 1.)

Tab. 1 only presents two ResNet models and their speedups are actually quite small. To see if the finetuning LR effect still holds across the full spectrum of pruning ratios and on other types of networks, we vary the pruning ratios from 0.1 to 0.95 and include experiments on VGG11_BN (Simonyan & Zisserman, 2015).

Results are presented in Tab. 2. With a more proper finetune LR scheme (column "Pruned-Fintuned-2" vs. "Pruned-Fintuned-1"), the performance can be improved *significantly*. A clear pattern is, the larger the pruning ratio, the more of the improvement. Now, comparing the results of "Pruned-Fintuned-2" to those of "Scratch", we can see pruning *outperforms* scratch-training in most cases. Exceptions appear on ResNet-34/18 under extreme pruning ratios (90% and 95%). Despite them, we believe it is fair to say **inheriting weights *has* value given the fact that $17/20$ experiments in Tabs. 1 and 2 show pruning is better than training from scratch**, especially under the pruning ratios of practical interest (*i.e.*, non-extreme pruning ratios). Retrospectively, (Liu et al., 2019) concluded oppositely because they faithfully re-implemented the L_1-norm pruning method *just according to the description in the original paper* (Li et al., 2017): fixed LR 10^{-3}, 20 epochs, which turns out *far from optimal* as we know now.

Reexamination of (Crowley et al., 2018). Coincidentally, (Crowley et al., 2018) adopted a *very similar* finetuning LR scheme to (Liu et al., 2019): They finetuned the pruned network with the lowest LR ($8 * 10^{-4}$, close to 10^{-3} in (Liu et al., 2019)) during scratch training and also *fixed*. Like the empirical study above, we reproduce the experiments of (Crowley et al., 2018) and rerun them with a larger initial LR (10^{-2}) and decay it during finetuning.

Detailed results are deferred to the Appendix (Tab. 7) due to the limited length here. We summarize the observation here – Exactly the same as the case in (Liu et al., 2019), when the proper finetuning LR is used, pruning actually *outperforms* the best scratch training scheme consistently.

Up to now, the results above have shown that the "no value of inheriting weights" argument in previous works is largely attributed to sup-optimal finetuning settings. A larger LR (*e.g.*, 10^{-2}) can significantly improve the finetuning performance than a small one (*e.g.*, 10^{-3}). In fact, we are not the only one to discover this. Previous works (Renda et al., 2020; Le & Hua, 2021) also reported similar observation. Nevertheless, they do not link the phenomenon with the "value of inheriting weights" argument and do not conduct systematical empirical studies as we do here. *More importantly, neither of them presented theoretical explanations about its occurrence* – next, we are about to bridge this gap. We present a faithful theoretical explanation through the lens of dynamical isometry.

3 Dynamical Isometry is the Key

3.1 Prerequisite: Dynamical Isometry

Dynamical isometry (DI) is studied under the topic of trainability of deep neural networks. It was first brought up in (Saxe et al., 2014). Specifically, the dynamical isometry is defined as *the singular values of the Jacobian matrix being around 1* (Saxe et al., 2014). It is easy to see that the networks with dynamical isometry is easy to train, because JSVs around 1 imply the gradient signals will not be amplified or attenuated seriously during propagation, preventing the network from gradient exploding or vanishing, which are well-known the main difficulties in deep network training (Glorot & Bengio, 2010; Sutskever et al., 2013). For linear networks, dynamical isometry can be achieved *exactly* by the orthogonal initialization proposed in (Saxe et al., 2014); while for neural networks with non-linearity (like ReLU (Nair & Hinton, 2010)) and convolution, it can only be approximated up to date (see Tab. 3).

Mean Jacobian singular values. Since dynamical isometry is measured by the Jacobian singular values (JSV's), we adopt *the mean of Jacobian singular values* (denoted by $\bar{S}$) as a scalar metric for analysis. Specifically, for a Jacobian $\mathbf{J} \in \mathbb{R}^{C \times D_{\text{in}}}$ (C stands for the output dimension, *i.e.*, the number of classes, D_{in} for the input dimension), apply singular value decomposition (Trefethen &

Table 3: JSVs (Jacobian singular values) of orthogonal initialization (Saxe et al., 2014) on different types of neural networks on MNIST dataset. Note, only the linear MLP network can achieve dynamical isometry *exactly* (*i.e.*, all the JSVs equal to 1).

Network	Mean JSV	Max JSV	Min JSV
MLP-7-Linear	$1.0000_{\pm 0.0000}$	1.0000	1.0000
MLP-7-ReLU	$1.2268_{\pm 0.5519}$	3.2772	0.2282
LeNet-5-Linear	$0.9983_{\pm 0.0842}$	1.2330	0.7896
LeNet-5-ReLU	$1.8331_{\pm 0.5731}$	3.6007	0.6151

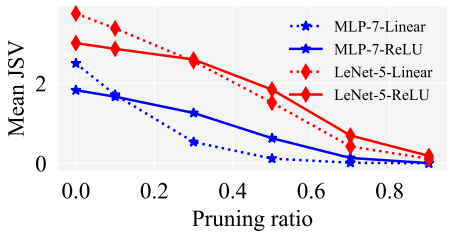


Figure 1: Mean JSV (Eq. (1)) of pruned networks w.r.t. different pruning ratios on the MNIST dataset. Note, with a larger pruning ratio, mean JSV is hurt more.

Bau III, 1997) to it,

$$U, \Sigma, V = \text{svd}(\mathbf{J}), \bar{S} = \frac{1}{K} \sum_{i=1}^{K} \Sigma_{ii}, \tag{1}$$

where Σ is the singular value matrix and $K = \min(C, D_{\text{in}})$.

Pruning as *poor* initialization. We investigate how the pruning affects $\bar{S}$. The results are shown in Fig. 1. As seen, the mean JSV is consistently *damaged* by pruning; and a larger pruning ratio, more decrease of the mean JSV. This means, pruning actually servers as a very *poor* initialization scheme for the subsequent finetuning.

In stark contrast to the broad awareness that initialization is rather critical to neural network training (Glorot & Bengio, 2010; Sutskever et al., 2013; Mishkin & Matas, 2016; Krähenbühl et al., 2015; He et al., 2015), the initialization role of pruning has received negligible research attention, however. As far as we know, no prior works have noted this issue when pruning a pretrained network or tried to recover the broken dynamical isometry before finetuning.

Simply put, our goal next is to show it is this broken dynamical isometry that answers for the performance gap between LR 10^{-2} and 10^{-3}. If dynamical isometry is fully recovered before finetuning, LR 10^{-2} vs. 10^{-3} should not cause significant performance difference anymore. To validate this explanation, we need a method to *fully recover* dynamical isometry in filter pruning, as is introduced next.

3.2 Dynamical Isometry Recovery in Filter Pruning

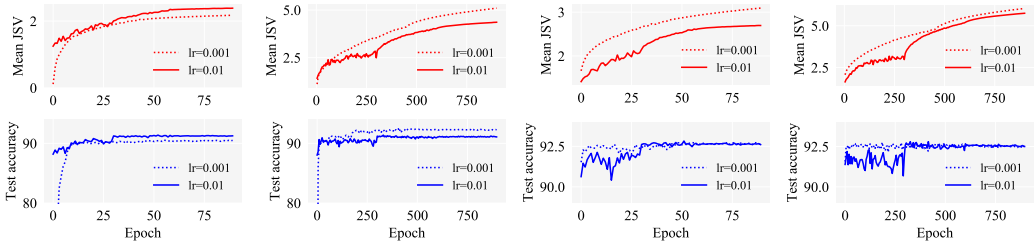
In (Saxe et al., 2014), they propose a weight orthogonalization scheme to achieve dynamical isometry for neural network *initialization*, namely, the initial weights are *randomly sampled*. Different from their case, here the initial weights are inherited from a pretrained model by pruning. Therefore, we need to adapt it to our application.

For a fully-connected layer parameterized by a matrix $W_0 \in \mathbb{R}^{J \times K}$ (for a convolutional layer parameterized by a 4-d tensor of shape $\mathbb{R}^{N \times C \times H \times W}$, it can be reshaped to a matrix of shape $\mathbb{R}^{N \times CHW}$), it reduces to matrix W of size $\mathbb{R}^{J_1 \times M_1}$ ($J_1 \leq J, K_1 \leq K$) after structured pruning. Then, we apply the weight orthogonalization technique (Mezzadri, 2006) based on QR decomposition (Trefethen & Bau III, 1997) to W,

$$\begin{aligned} Q, R &= \text{qrd}(W), \\ W^* &= Q \odot \text{sign}(\text{diag}(R)), \end{aligned} \tag{2}$$

where qrd$(\cdot)$ means QR decomposition; Q is an orthogonal matrix of the same size as W ($\mathbb{R}^{J_1 \times K_1}$); R is an an upper triangular matrix of size $\mathbb{R}^{K_1 \times K_1}$; sign$(\cdot)$ is the sign function which returns the positive or negative sign of its argument; $\odot$ represents the Hadamard (element-wise) product aligned to the last axis (since Q and sign(diag(R)) share the same dimension at the last axis).

As an orthogonalized version of W, W^* recovers the dynamical isometry damaged by pruning. Therefore, we propose to employ W^* instead of the original W as the initialization weights for later finetuning. We dub this weight orthogonalization method for *pruned* models as **OrthP**. With this method, we can fully recover broken dynamical isometry and continue our analysis as follows.



(a) 90 epochs (b) 900 epochs (c) OrthP, 90 epochs (d) OrthP, 900 epochs

Figure 2: Mean JSV and test accuracy during finetuning with different setups. Note that with OrthP (c, d), mean JSV recovers faster; so does the test accuracy. The pruning ratio in this case is 0.8 (see Appendix Fig. 3 for similar plots of pruning ratio 0.9).

3.3 Analysis with MLP-7-Linear on MNIST

Evaluated network. The network for analysis is a 7-layer linear MLP. We are aware that this toy network has little practical meaning, but it is very appropriate here for two reasons. First, as mentioned above, in our analysis we need a method to recover DI *exactly*. Up to date, this can *only* be achieved on linear networks (see Tab. 3). Second, the linear MLP network is free from the intervention of modern CNN features (*e.g.*, BN (Ioffe & Szegedy, 2015), residual (He et al., 2016)). By our observation, these features will make the problem complex and prevent us from seeing consistent results at the early analysis stage.

LR schedule setup. When we set different LR schedules with different initial LRs, we will (1) keep the total number of epochs the same, (2) keep the last LR the same, (3) intentionally use *prolonged* training epochs (like 90 or even 900 epochs on MNIST dataset. Typically, it only needs 30 epochs to reach convergence on MNIST). All of these are to ensure the networks are fully converged, helping us render a faithful conclusion. Step decay LR schedule is employed given its broad use.

Proposed explanation and its deducted hypotheses. We first plot the mean JSV during finetuning of the pruned MLP-7-Linear network in Fig. 2. Notably, one important fact from Fig. 2 is that without any extra help, the mean JSV can recover itself during finetuning (see Row 1). Aware of this, it is straightforward to conjecture that any factor (*e.g.*, learning rate) involved in the training can affect dynamical isometry. This inspires us to the following plausible explanation about how a larger finetuning LR can improve performance significantly:

> *A larger finetuning LR helps the network converge faster. Thus the dynamical isometry (measured by mean JSV) recovers faster and better. Since better dynamical isometry promotes faster and better convergence, a larger LR leads to the better final performance.*

That is, a larger finetuning LR shows performance advantage *only if dynamical isometry is broken by pruning first*. If dynamical isometry is fully recovered before finetuning, a larger LR should *not* pose performance advantage anymore. The validation of this explanation can be specified into the following 4 deducted hypotheses:

- **Hypothesis 1**: Given a small number of epochs, mean JSV *cannot* be fully recovered by training, then the larger LR should show a significant advantage over the smaller LR.
- **Hypothesis 2**: With sufficient training epochs, mean JSV can be fully recovered by training. Then the larger LR should have less advantage over the smaller LR.
- **Hypothesis 3**: If we employ OrthP to exactly recover the mean JSV, given the small number of epochs again, the larger LR should have much less advantage now.
- **Hypothesis 4**: If we combine abundant epochs with OrthP, mean JSV will be recovered even completely, then the performance advantage of the larger LR over the smaller one should be even weaker.

Corresponding to these four hypotheses, the eight finetuning LR settings are summarized in Tab. 4. The unpruned MLP model is trained with LR schedule "90 epochs, initial 10^{-2}, decay 30/60". For filter pruning, we employ L_1-norm pruning (Li et al., 2017) throughout this paper. Specifically, it

Table 4: Summary of finetuning LR setups corresponding to the 4 proposed hypotheses in Sec. 3.3.

	Initial LR 10^{-2}	Initial LR 10^{-3}
For Hypothesis 1	90 epochs, Initial 10^{-2}, decay 30/60	90 epochs, Initial 10^{-3}, decay 45
For Hypothesis 2	900 epochs, Initial 10^{-2}, decay 300/600	900 epochs, Initial 10^{-3}, decay 450
For Hypothesis 3	OrthP, 90 epochs, Initial 10^{-2}, decay 30/60	OrthP, 90 epochs, Initial 10^{-3}, decay 45
For Hypothesis 4	OrthP, 900 epochs, Initial 10^{-2}, decay 300/600	OrthP, 900 epochs, Initial 10^{-3}, decay 450

Table 5: Test accuracies (%) of the 4 hypotheses in Tab. 4 with MLP-7-Linear network on MNIST. Accuracy of unpruned model: 92.77%. Each setting is randomly run 5 times, mean accuracy and stddev reported. "Acc. gain" refers to the mean accuracy improvement of LR 10^{-2} over 10^{-3}. Notably, whether LR 10^{-2} is better than 10^{-3} actually depends on the specific finetuning setting, fundamentally caused by the damaged dynamical isometry. Without knowing this, the comparison results can lead us to partial conclusion even they appear fair (*e.g.*, the "90 epochs" row).

Finetuning setting	Pruning ratio 80%			Pruning ratio 90%		
	LR 10^{-3}	LR 10^{-2}	Acc. gain	LR 10^{-3}	LR 10^{-2}	Acc. gain
90 epochs	$90.54_{\pm 0.02}$	$\mathbf{91.36}_{\pm \mathbf{0.02}}$	0.82	$87.59_{\pm 0.01}$	$\mathbf{87.81}_{\pm \mathbf{0.03}}$	0.22
900 epochs	$\mathbf{92.54}_{\pm \mathbf{0.03}}$	$91.64_{\pm 0.41}$	−0.90	$\mathbf{90.44}_{\pm \mathbf{0.01}}$	$87.83_{\pm 0.04}$	−2.61
OrthP, 90 epochs	$92.77_{\pm 0.03}$	$\mathbf{92.79}_{\pm \mathbf{0.03}}$	0.02	$92.72_{\pm 0.03}$	$\mathbf{92.77}_{\pm \mathbf{0.04}}$	0.05
OrthP, 900 epochs	$\mathbf{92.84}_{\pm \mathbf{0.03}}$	$92.81_{\pm 0.04}$	−0.03	$\mathbf{92.86}_{\pm \mathbf{0.03}}$	$92.79_{\pm 0.03}$	−0.07

sorts the neurons (or filters) by their L_1 norms in ascending order and prunes those with the least norms by a predefined pruning ratio r.

The final accuracy results are shown in Tab. 5, training plots shown in Fig. 2. We first analyze the results of pruning ratio 0.8 in Tab. 5. As seen, when finetuned for 90 epochs, LR 10^{-2} shows an advantage over LR 10^{-3} by 0.82% accuracy. It is tempting to draw a conclusion based on this comparison that LR 10^{-2} is much better than LR 10^{-3}. However, this is not the whole story:

- With 900 epochs, LR 10^{-2} is greatly surpassed by LR 10^{-3} (91.64 vs. 92.54). The reason by our proposed explanation is that, with abundant epochs, the dynamical isometry can be recovered more completely, hence LR 10^{-2} does not show advantages anymore over 10^{-3}.
- When OrthP applied, LR 10^{-2} does not show significant advantages either, similar to the effect of increasing the number of training epochs. This is because that finetuning shares the same role of recovering dynamical isometry with OrthP. Just OrthP is more effective since it is analytically targeting exact dynamical isometry.
- When the best setting used (OrthP + 900 epochs), LR 10^{-3} is slightly better than 10^{-2}. Comparing "OrthP, 900 epochs" with "OrthP, 90 epochs", the gains are only marginal. This is because the dynamical isometry has already been fully recovered by OrthP, thus more training epochs do not show much value anymore.

A different pruning ratio 0.9 is also explored. Its results are in line with those of pruning ratio 0.8 as shown in Tab. 5. In short, these empirical observations are fully in line with our expectations, justifying the validity of the proposed explanation.

Explaining LR effect on ImageNet. In Tab. 2, there is an apparent trend that the *larger* pruning ratio, the *more* performance advantage of LR 10^{-2} over 10^{-3} (this also appears on CIFAR10 dataset, see Tab. 6). Using our explanation, this phenomenon can also be explained now – When the pruning ratio is greater, more dynamical isometry is damaged. LR 10^{-2} can find more use in these cases since it is faster/better to recover dynamical isometry, hence the more pronounced advantage.

A closer look and more lessons. In the ResNet34 results above (Tab. 1) reported by (Li et al., 2017) and (Liu et al., 2019)), it is still easy to notice their finetuning LR schedule may be sub-optimal given their finetuning process (20 epochs) is obviously shorter than training from scratch (90 epochs). For Tab. 5, quite differently, the "90 epochs" finetuning setting appears *nothing wrong* (considering the unpruned network is trained for 90 epochs, finetuning for 90 epochs is not short; besides, the LR is also decayed to a very small amount). However, the results of this setting actually lead us to a partial conclusion that the larger LR is better than the smaller LR. This particular example shows *how misleading the comparison results can be, even though they look perfectly fair, if we are not aware of the effect of dynamical isometry in structured pruning*.

Table 6: Test accuracies (%) of pruning ResNet56 on CIFAR10. Unpruned accuracy: 93.78%. Each setting is randomly run 3 times. "Acc. gain" means the accuracy gain of initial LR 10^{-2} over 10^{-3}.

Pruning ratio r	0.5	0.7	0.9	0.95
Sparsity/Speedup	49.82%/1.99×	70.57%/3.59×	90.39%/11.41×	95.19%/19.31×
	Finetuning LR schedule: 120 epochs, initial 10^{-2}, decay 60/90			
Train from scratch	$92.78_{\pm 0.23}$	$92.11_{\pm 0.12}$	$88.36_{\pm 0.20}$	$84.60_{\pm 0.14}$
L_1 (Li et al., 2017)	$93.51_{\pm 0.07}$	$92.26_{\pm 0.17}$	$88.71_{\pm 0.15}$	$84.63_{\pm 0.28}$
L_1 + OrthP	$93.36_{\pm 0.19}$	$91.96_{\pm 0.06}$	$86.01_{\pm 0.34}$	$82.62_{\pm 0.05}$
StrongReg	$\mathbf{93.55_{\pm 0.06}}$	$\mathbf{92.38_{\pm 0.09}}$	$\mathbf{89.24_{\pm 0.16}}$	$\mathbf{85.90_{\pm 0.19}}$
	Finetuning LR schedule: 120 epochs, initial 10^{-3}, decay 80			
L_1 (Li et al., 2017)	$93.12_{\pm 0.10}$	$91.77_{\pm 0.11}$	$87.57_{\pm 0.09}$	$83.10_{\pm 0.12}$
StrongReg	$\mathbf{93.44_{\pm 0.06}}$	$\mathbf{91.96_{\pm 0.11}}$	$\mathbf{88.69_{\pm 0.19}}$	$\mathbf{85.45_{\pm 0.25}}$
Acc. gain. (L_1)	0.39	0.49	1.14	1.53
Acc. gain. (StrongReg)	**0.11**	**0.42**	**0.55**	**0.45**

4 Dynamical Isometry Recovery (DIR)

By our analysis above, dynamical isometry *recovery* before finetuning is rather important. An effective dynamical isometry recovery method is supposed to (1) close up the performance gap between finetuning LR 10^{-2} and 10^{-3}, (2) improve the final performance (otherwise, there is no point using it at all). As shown in Tab. 5, the proposed OrthP method for linear MLP networks can meet these two requirements; and since it recovers DI exactly, it actually completely closes the performance gap between finetuning LR 10^{-2} and 10^{-3}. Unfortunately, this method does not generalize to more practical non-linear convolutional neural networks, as shown in Tab. 6, where using OrthP *degrades* the final performance. As a remedy, in this section we introduce a very simple regularization-based method that can work on practical CNNs.

Strong L_2 regularization as a drop-in remedy. In a typical structured pruning algorithm, unimportant filters are selected by some criterion. Then they are removed (zeroed out), followed by a finetuning process. We propose to apply a *super strong* L_2 regularization (*e.g.*, regularization factor equals to 1) to push the unimportant filters to rather close to zero first, before permanently removing them. This simple technique can readily work as a drop-in step into *any* structured pruning algorithm and it is very easy to implement on any deep learning framework.

Results in Tab. 6 demonstrate the effectiveness of this simple technique on ResNet56. It improves the final pruning performance and also shrinks the performance gap between finetuning LR 10^{-2} and 10^{-3} (especially when the pruning ratio is large), meeting the two requirements above. We are aware that this technique is preceded by several variants in the past literature (*e.g.*, GReg-1 in (Wang et al., 2021b)), but they do not link their method with dynamical isometry recovery as we do here. Also, our technique is even simpler than theirs – in (Wang et al., 2021b), L_2 regularization is *increased gradually* to a very strong level, while here we use a *fixed* strong regularization, which probably is the *simplest* form of utilizing a strong regularization.

5 Conclusion

In this work, we present extensive empirical evidences to show the "no value of inheriting weights" argument in prior works is inaccurate because of improper finetuning LR schedules. We further tap into dynamical isometry to explain why the finetuning LR has such a great impact on the final performance, through carefully designed control experiments with 4 hypotheses. For practically amending the broken dynamical isometry, we also present a rather simple regularization-based technique that works effectively on residual convolutional networks.

The finding of dynamical isometry in structured pruning in this paper justifies the value of inheriting weights, in line with the past research wisdom and our common beliefs. It also helps us towards a better understanding of pruning and possibly can inspire more advanced pruning algorithms as dynamical isometry recovery has been shown a worthy direction in the paper. In addition, the awareness of dynamical isometry in structured pruning can help us render a more faithful conclusion when comparing different pruning methods.

REFERENCES

Davis Blalock, Jose Javier Gonzalez, Jonathan Frankle, and John V Guttag. What is the state of neural network pruning? In *SysML*, 2020. 3

Jian Cheng, Pei-song Wang, Gang Li, Qing-hao Hu, and Han-qing Lu. Recent advances in efficient computation of deep convolutional neural networks. *Frontiers of Information Technology & Electronic Engineering*, 19(1):64–77, 2018a. 3

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35 (1):126–136, 2018b. 3

Elliot J Crowley, Jack Turner, Amos Storkey, and Michael O'Boyle. A closer look at structured pruning for neural network compression. *arXiv preprint arXiv:1810.04622*, 2018. 1, 3, 5, 12

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 1, 3

Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4): 485–532, 2020. 3

Xiaohan Ding, Guiguang Ding, Jungong Han, and Sheng Tang. Auto-balanced filter pruning for efficient convolutional neural networks. In *AAAI*, 2018. 2

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *JMLR*, 20(55):1–21, 2019. 3

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019. 3

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *ICML*, 2020. 3

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark? In *ICLR*, 2021. 3

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019. 3

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010. 5, 6

Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015. 2, 3

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016. 2, 5

B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *NeurIPS*, 1993. 2

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 7

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *CVPR*, 2015. 6

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017. 2, 3, 5

Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *ECCV*, 2018. 3

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 7

Philipp Krähenbühl, Carl Doersch, Jeff Donahue, and Trevor Darrell. Data-dependent initializations of convolutional neural networks. *arXiv preprint arXiv:1511.06856*, 2015. 6

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 3

Duong H Le and Binh-Son Hua. Network pruning that matters: A case study on retraining variants. In *ICLR*, 2021. 1, 5

Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *NeurIPS*, 1990. 1, 2

Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 3

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. Snip: Single-shot network pruning based on connection sensitivity. In *ICLR*, 2019. 3

Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip HS Torr. A signal propagation perspective for pruning neural networks at initialization. In *ICLR*, 2020. 3

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, 2017. 2, 3, 4, 5, 7, 8, 9, 12, 14

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017. 3

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *ICLR*, 2019. 1, 2, 3, 4, 5, 8, 14

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017. 3

Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J Dally. Exploring the granularity of sparsity in convolutional neural networks. In *CVPR Workshop*, 2017. 2

Francesco Mezzadri. How to generate random matrices from the classical compact groups. *arXiv preprint math-ph/0609050*, 2006. 6

Dmytro Mishkin and Jiri Matas. All you need is a good init. In *ICLR*, 2016. 6

Michael C Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *NeurIPS*, 1989. 1

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010. 5

Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *CVPR*, 2020. 3

R. Reed. Pruning algorithms – a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993. 1, 3

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *ICLR*, 2020. 1, 5

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *ICLR*, 2014. 2, 5, 6

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 5

Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximations for model compression. In *NeurIPS*, 2020. 2

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013. 5, 6

Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017. 1, 3

Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018. 2, 12

Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997. 5, 6

Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. Eigendamage: Structured pruning in the kronecker-factored eigenbasis. In *ICML*, 2019a. 2

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *ICLR*, 2020. 3

Huan Wang, Qiming Zhang, Yuehai Wang, Lu Yu, and Haoji Hu. Structured pruning for efficient convnets via incremental regularization. In *IJCNN*, 2019b. 2

Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Emerging paradigms of neural network pruning. *arXiv preprint arXiv:2103.06460*, 2021a. 3

Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. In *ICLR*, 2021b. 2, 9

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *NeurIPS*, 2016. 2, 5

Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 3

A RESULTS OF REEXAMINATION OF (CROWLEY ET AL., 2018)

Table 7: Test accuracy comparison between 2 pruning schemes and 4 scratch training schemes in (Crowley et al., 2018). Network: **WRN-40-2** (unpruned accuracy: 95.08%, params: 2.24M). Dataset: **CIFAR-10**. Results above the dashline are directly cited from (Crowley et al., 2018); results below the dashline are from our reproducing (with the official code of (Crowley et al., 2018) at https://github.com/BayesWatch/pytorch-prunes for fair comparison). "Rerun" means we rerun the code of (Crowley et al., 2018) as it is. "LR 10^{-2}" means we redo the finetuing for the pruned models in "Rerun" *using our finetuning LR schedule* (120 epochs, initial 10^{-2}, decay 60/90). Finetuning is randomly repeated for 3 times, mean (stddev) accuracies reported. The main point here is that (Crowley et al., 2018) draws the conclusion that scratch training is better than pruning *because of an improper finetuning LR scheme*. With the proper finetuning LR scheme, pruning is actually *better* than scratch training.

Method	Scratch?	500K params budget		1M params budget		1.5M params budget	
		Params (M)	Acc. (%)	Params (M)	Acc. (%)	Params (M)	Acc. (%)
L_1-norm pruning (Li et al., 2017)	✗	0.51	90.86	1.02	92.61	1.52	93.63
Fisher pruning (Theis et al., 2018)	✗	0.52	92.59	1.02	93.51	1.52	94.51
Varying Depth	✓	0.69	93.56	1.08	94.54	1.47	94.64
Varying Width	✓	0.50	93.45	0.98	94.30	1.48	94.66
Varying Bottleneck	✓	0.50	93.69	1.00	94.40	1.49	94.79
Fisher Scratch	✓	0.52	**93.72**	1.02	**94.65**	1.52	**94.86**
L_1-norm pruning (Li et al., 2017) (Rerun)	✗	0.50	91.23	1.00	92.80	1.51	93.52
L_1-norm pruning (Li et al., 2017) (LR 10^{-2})	✗	0.50	93.88 (0.10)	1.00	94.49 (0.10)	1.51	94.92 (0.16)
Fisher pruning (Theis et al., 2018) (Rerun)	✗	0.52	92.17	0.98	93.57	1.48	94.67
Fisher pruning (Theis et al., 2018) (LR 10^{-2})	✗	0.52	**94.27** (0.09)	0.98	**94.80** (0.02)	1.48	**95.10** (0.13)

Table 8: Training setting summary in finetuning. For the SGD solver, in the parentheses are the momentum and weight decay.

Dataset	MNIST	CIFAR10	ImageNet
Solver	SGD (0.9, 5e-4)	SGD (0.9, 5e-4)	SGD (0.9, 1e-4)
LR schedule (Initial 1e-2)	Multi-step (decay 30/60)	Multi-step (decay 60/90)	Multi-step (decay 30/60/75)
LR schedule (Initial 1e-3)	Multi-step (decay 45)	Multi-step (decay 80)	Multi-step (decay 45/68)
Total epoch	90	120	90
Batch size	100	128	256
Data augmentation	None	Random crop and horizontal flip	Random crop and horizontal flip

B Training Setting Summary

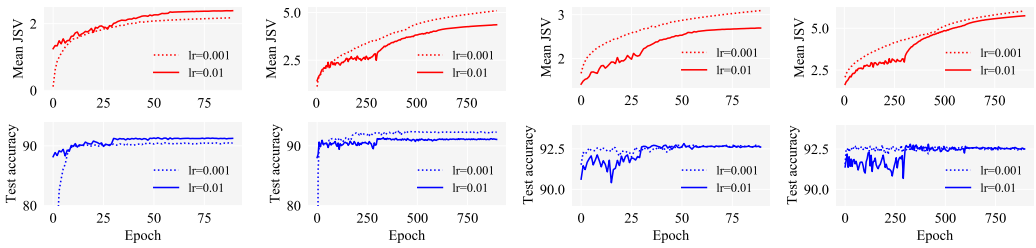
There are three datasets in our experiments in the paper: MNIST, CIFAR10, and ImageNet. Apart from some key settings stated in the paper, a more detailed training setting summary is shown as Tab. 8. We use 8 NVIDIA V100 GPUs for all our experiments.

A typical finetuning LR schedule looks like this in our paper: "90 epochs, initial 10^{-2}, decay 30/60". Its meaning is: the total number of training epochs is 90; initial LR is 10^{-2}; at epoch 30 and 60, LR is multiplied by a default factor 0.1. The others can be inferred likewise.

C More JSV and Accuracy Plots

In Fig. 2 of the main paper, we present the training plots of pruning ratio 0.8. Here we show the case of pruning ratio 0.9 in Fig. 3. The general observations are very similar to those of the pruning ratio 0.8:

- Mean JSV can recover itself during finetuning across all the 4 settings.
- Without OrthP (see (a) and (b)), mean JSV starts lower than that in (c) and (d). LR 0.01 helps the JSV recover faster than LR 0.001 so the red *solid* line starts higher than the red *dotted* line. So does the test accuracy. For (a), the number of epochs is limited, thus over the whole finetuning, mean JSV of LR 0.01 is always above that of LR 0.001. In contrast, for (b), when the mean JSV is fully recovered by abundant epochs, LR 0.001 starts to take lead and gets beyond LR 0.001; note the test accuracy lines also present the same trend, in accordance with the trend of mean JSV plots. It shows that mean JSV has a lot to do with the test accuracy – *a higher mean JSV implies a better test accuracy*.
- With OrthP (see (c) and (d)), mean JSV is explicitly recovered by OrthP, thus the red lines start higher than those in (a) and (b). So do the test accuracy lines (after the 1st epoch, the test accuracies reach about 90%, as compared to less than 75% in (a) and (b)). This shows JSV recovery is beneficial to faster convergence, further validating one of our claims in the paper: dynamical isometry recovery before finetuning is necessary.



(a) 90 epochs (b) 900 epochs (c) OrthP, 90 epochs (d) OrthP, 900 epochs

Figure 3: Mean JSV and test accuracy during finetuning with different setups. Note that with OrthP (c, d), mean JSV recovers faster; so does the test accuracy. The pruning ratio in this case is 0.9. Similar plots of pruning ratio 0.8 are shown in Fig. 2 in the main paper. (*This figure is best viewed in color*).

D WHY OUR REPRODUCED RESULTS ARE BETTER THAN (LIU ET AL., 2019)

In Tab. 1, we present the results of ResNet34 on ImageNet, pruned by the L_1-norm pruning method (Li et al., 2017). Readers may be curious that *why our scratch-training results are much higher than those of Rethinking (Liu et al., 2019) (by* 0.4-0.6%*).* This has a historical reason. (Liu et al., 2019)[2] refers to the official PyTorch ImageNet example[3], where the LR is only decayed *twice* (from 1e-1 to 1e-2, then to 1e-3). However, by our empirical observation, 1e-3 is still not the lowest LR that the network converges. If the LR is decayed another time (to 1e-4), the top-1 accuracy can still bump by round 0.5-0.8% point, which is a significant improvement for ImageNet thus not negligible. Therefore, we choose to decay the LR further to 1e-4 and finally to 1e-5 to ensure the network is *fully converged*. This reiterates the comparison rule of our paper: comparing different methods *in their best shape*. Comparing scratch training to L_1-norm pruning *before* the finetuned network finally converges may appear fair (given the same number of epochs) but is of less practical meaning and may hide the true picture.

[2]https://github.com/Eric-mingjie/rethinking-network-pruning/tree/master/imagenet/l1-norm-pruning
[3]https://github.com/pytorch/examples/tree/master/imagenet