Gradient Flow in Sparse Neural Networks and How Lottery Tickets Win

Utku Evci* Yani A. Ioannou* Cem Keskin Yann Dauphin
Google
{evcu,yai,cemkeskin,ynd}@google.com

October 8, 2020

Abstract

Sparse Neural Networks (NNs) can match the generalization of dense NNs using a fraction of the compute/storage for inference, and have the potential to enable efficient training. However, naively training unstructured sparse NNs from random initialization results in significantly worse generalization, with the notable exceptions of Lottery Tickets (LTs) and Dynamic Sparse Training (DST). In this work, we attempt to answer: (1) why training unstructured sparse networks from random initialization performs poorly and; (2) what makes LTs and DST the exceptions? We show that sparse NNs have poor gradient flow *at initialization* and propose a modified initialization for unstructured connectivity. Furthermore, we find that DST methods significantly improve gradient flow *during training* over traditional sparse training methods. Finally, we show that LTs do not improve gradient flow, rather their success lies in re-learning the pruning solution they are derived from — however, this comes at the cost of learning novel solutions.

1 Introduction

Deep Neural Networks (DNNs) are the state-of-the-art method for solving problems in computer vision, speech recognition, and many other fields. While early research in deep learning focused on application to new problems, or pushing state-of-the-art performance with ever larger/more computationally expensive models, a broader focus has emerged towards their efficient real-world application. One such focus is on the observation that only a sparse subset of this dense connectivity is required for inference, as apparent in the success of *pruning* (Han et al., 2015; Mozer et al., 1989b).

Pruning has a long history in Neural Network (NN) literature, and remains the most popular approach for finding sparse NNs. Sparse NNs found by pruning algorithms (Han et al., 2015; Louizos et al., 2017; Molchanov et al., 2017; Zhu et al., 2018) (i.e. *pruning solutions*) can match dense NN generalization with much better efficiency at *inference* time. However, naively *training* an (unstructured) sparse NN from a random initialization (i.e. *from scratch*), typically leads to significantly worse generalization.

Two methods in particular have shown some success at addressing this problem — Lottery Tickets (LTs) and Dynamic Sparse Training (DST). The mechanism behind the success of both of these methods is not well understood however, e.g. we don't know how to find Lottery Tickets (LTs) efficiently; while RigL (Evci et al., 2020), a recent DST method, requires $5\times$ the training steps to match dense NN generalization. Only in understanding how these methods overcome the difficulty of sparse training can we improve upon them.

A significant breakthrough in training DNNs — addressing vanishing and exploding gradients — arose from understanding gradient flow both at initialization, and during training. In this work we investigate the role of gradient flow in the difficulty of training unstructured sparse NNs from random initializations and from LT initializations. Our experimental investigation results in the following insights:

*These authors contributed equally to this paper.

1. **Sparse NNs have poor gradient flow at initialization.** In §3.1, §4.1 we show that existing methods for initializing sparse NNs are incorrect in not considering heterogeneous connectivity. We believe we are the first to show that sparsity-aware initialization methods improve gradient flow and training.

2. **Sparse NNs have poor gradient flow during training.** In §3.2, §4.2, we observe that even in sparse NN architectures less sensitive to incorrect initialization, the gradient flow *during training* is poor. We show that DST methods achieving the best generalization have improved gradient flow.

3. **Lottery Tickets don't improve upon (1) or (2), instead they re-learn the pruning solution.** In §3.3, §4.3 we show that a LT initialization resides within the same basin of attraction as the original pruning solution it is derived of, and a LT solution is highly similar to the pruning solution in function space.

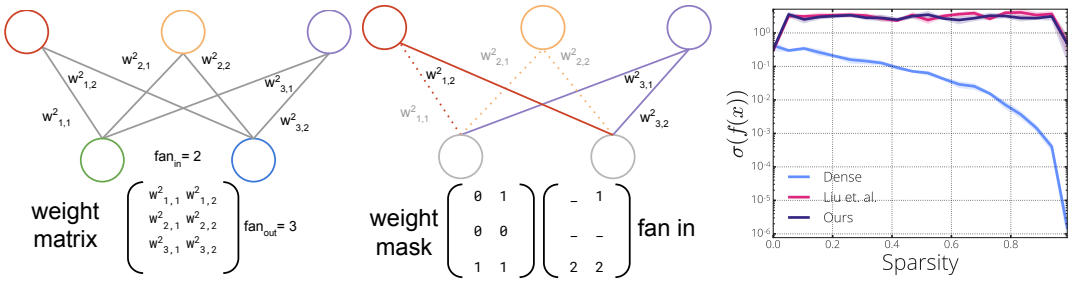
2 Related Work

Pruning Pruning is used commonly in Neural Network (NN) literature to obtain sparse networks (Castellano et al., 1997; Hanson et al., 1988; Kusupati et al., 2020; Mozer et al., 1989a,b; Setiono, 1997; Sietsma et al., 1988; Wortsman et al., 2019). Pruning algorithms remove connections of a trained dense network using various criteria including weight magnitude (Han et al., 2015, 2016; Zhu et al., 2018), gradient-based measures (Molchanov et al., 2016), and 2^{nd}-order terms based on the Hessian (Hassibi et al., 1993; LeCun et al., 1990). While the majority of pruning algorithms focus on pruning after training, a subset focuses on pruning NNs before training (Lee et al., 2019; Tanaka et al., 2020; Wang et al., 2020). Gradient Signal Preservation (GRaSP) (Wang et al., 2020) is particularly relevant to our study, since their pruning criteria aims to preserve gradient flow, and they observe a positive correlation between initial gradient flow and final generalization. However, recent work of Frankle et al., 2020c suggests that the reported gains are due to sparsity distributions discovered rather than the particular sub-network. Another limitation of these algorithms is that they don't scale to large scale tasks like Resnet-50 training on ImageNet-2012.

Lottery Tickets Frankle et al. (2019a) showed the existence of sparse sub-networks at initialization — known as Lottery Tickets — which can be trained to match the generalization of the corresponding dense Deep Neural Network (DNN). The initial work of Frankle et al. (2019a) inspired much follow-up work. Gale et al. (2019) and Liu et al. (2019) observed that the initial formulation was not applicable to larger networks with higher learning rates. Frankle et al. (2019b, 2020b) proposed *late rewinding* as a solution. Morcos et al. (2019) and Sabatelli et al. (2020) showed that Lottery Tickets (LTs) trained on large datasets transfer to smaller ones, but not *vice versa*. Frankle et al. (2020a), Ramanujan et al. (2019), and Zhou et al. (2019) focused on further understanding LTs, and finding sparse sub-networks at initialization. As one might expect, sufficiently large networks would have smaller solutions hidden in them. Malach et al. (2020) studied this and proved the existence of solutions in sufficiently large networks. However, it is an open question whether finding such networks at initialization could be done more efficiently than with existing pruning algorithms.

Dynamic Sparse Training Most training algorithms work on pre-determined architectures and optimize parameters using fixed learning schedules. Dynamic Sparse Training (DST), on the other hand, aims to optimize the sparse NN connectivity jointly with model parameters. Mocanu et al. (2018) and Mostafa et al. (2019) propose replacing low magnitude parameters with random connections and report improved generalization. Dettmers et al. (2019) proposed using momentum values, whereas Evci et al. (2020) used gradient estimates directly to guide the selection of new connections, reporting results that are on par with pruning algorithms. In §4.2 we study these algorithms and try to understand the role of gradient flow in their success.

Random Initialization of Sparse NN In training sparse NN from scratch, the vast majority of pre-exisiting work on training sparse NN has used the common initialization methods (Glorot et al., 2010; He et al., 2015) derived for *dense* NNs, with only a few notable exceptions. Gale et al. (2019), Liu et al. (2019), and Ramanujan et al. (2019) scaled the variance (fan-in/fan-out)



(a) Dense Layer (b) Sparse Layer (c) Signal Propagation at Init.

Figure 1: **Glorot/He Initialization for a Sparse NN.** All neurons in a dense NN layer (a) have the same fan-in, whereas in a sparse NN (b) the fan-in can differ for *every neuron*, potentially requiring sampling from a different distribution for every neuron. The initialization derivation/fan-out variant are explained further in Appendix A.1. (c) Std. dev. of the pre-softmax output of LeNet5 with input sampled from a normal distribution, over 5 different randomly-initialized sparse NN for a range of sparsities.

of a sparse NN layer according to the layer's sparsity, effectively using the standard initialization for a small dense layer of equivalent number of weights as in the sparse model.

3 Analyzing Gradient Flow in Sparse Neural Networks

A significant breakthrough in training very deep NNs arose in addressing the *vanishing* and *exploding gradient* problem, both at initialization, and during training. This problem was understood by analyzing the signal propagation within a DNN, and addressed in improved initialization methods (Glorot et al., 2010; He et al., 2015; Xiao et al., 2018) alongside normalization methods, such as Batch Normalization (BatchNorm) (Ioffe et al., 2015). In our work, following Wang et al. (2020), we study these problems using the gradient flow, $\nabla L(\theta)^T \nabla L(\theta)$ which is the first order approximation* of the decrease in the loss expected after a gradient step. We observe poor gradient flow for the predominant sparse NN initialization strategy and propose a solution in §3.1. Then in §3.2 and §3.3 we summarize Dynamic Sparse Training (DST) methods and LT hypothesis respectively.

3.1 The Initialization Problem in Sparse Networks

Here we analyze the gradient flow at initialization for random sparse NNs, motivating the derivation of a more general initialization for NN with heterogeneous connectivity, such as in sparse NNs. In practice, without a method such as BatchNorm (Ioffe et al., 2015), using the correct initialization can be the difference between being able to train a DNN, or not — as observed for VGG16 in our results (§4.1, Table 1). The initializations proposed by Glorot et al. (2010) and He et al. (2015) ensure that the output distribution of every neuron in a layer is of zero-mean and unit variance, and do this by sampling a Gaussian distribution with a variance based on the number of incoming/outgoing connections for all the neurons in a dense layer, as illustrated in Fig. 1a, which is assumed to be identical for all neurons in the layer.

In an unstructured sparse NN however, the number of incoming/outgoing connections is not identical for all neurons in a layer, as illustrated in Fig. 1b. In Appendix A.2 we derive the initialization for this more general case. In Appendix A.1 we explain in full the generalized Glorot et al. (2010) and He et al. (2015) initialization, in the forward, backward and average use cases. Here we will focus only on explaining the generalized He et al. (2015) initialization for forward propagation, which we used in our experiments.

For every weight $w_{ij}^{[\ell]} \in W^{n^{[\ell]} \times n^{[\ell-1]}}$ in a layer ℓ with $n^{[\ell]}$ neurons, and mask $[m_{ij}^{[\ell]}] = M^{\ell} \in$

*We omit learning rate for simplicity.

$[0,1]^{n^{[\ell]} \times n^{[\ell-1]}}$,

$$w_{ij}^{[\ell]} \sim \mathcal{N}\left(0, \frac{2}{\textit{fan-in}_i^{[\ell]}}\right), \qquad \text{where } \textit{fan-in}_i^{[\ell]} = \sum_{j=1}^{n^{[\ell-1]}} m_{ij}^{[\ell]}, \tag{1}$$

is the number of incoming connections for neuron i in layer ℓ.

In the special case of a dense layer where $m_{ij}^{[\ell]} = 1, \forall i,j$, Eq. (1) reduces to the initialization proposed by (He et al., 2015) since $\textit{fan-in}_i^{[\ell]} = n^{[\ell-1]}, \forall i$. Using the dense initialization in a sparse DNN causes signal to vanish, as empirically observed in Fig. 1c), whereas our initialization keeps the variance of the signal constant. The initialization proposed by Liu et al. (2019) is a special case of ours where it is assumed $\textit{fan-in}_i^{[\ell]} \equiv \textit{fan-in}^{[\ell]}, \forall i$, i.e. all neurons have the same number of unmasked incoming connections in a layer. Surprisingly the initialization of Liu et al. (2019) also preserves the signal in Fig. 1c (discussed in §4.1).

3.2 Gradient Flow during Training and Dynamic Sparse Training

While initialization is important for the first training step, the gradient flow during the early stages of training is not well addressed by initialization alone, as shown by normalization methods (Ioffe et al., 2015). Our findings show that even with BatchNorm, the gradient flow during training in unstructured sparse NNs is poor.

Recently, a promising new approach to training sparse NNs has emerged — Dynamic Sparse Training (DST) — that learns connectivity adaptively during training, showing significant improvements over baseline methods that use a fixed mask. These methods perform periodic updates on the sparse connectivity of each layer: commonly replacing least magnitude connections with new connections selected using various criteria. We consider two of these methods: Sparse Evolutionary Training (SET) (Mocanu et al., 2018), which chooses new connections randomly and Rigged Lottery (RigL) (Evci et al., 2019), which chooses connections with high gradient magnitude. RigL improves over SET and matches pruning performance with sufficient training time. Since these methods have only recently been proposed, there is a lack of understanding of why and how these methods achieve better results.

3.3 Lottery Ticket Hypothesis

A recent approach for training unstructured sparse NNs while achieving similar generalization to the original dense solution is the Lottery Ticket Hypothesis (LTH) (Frankle et al., 2019a). Notably, rather than training a pruned NN structure from random initialization, the LTH uses the dense initialization from which the pruning solution was trained/derived from.

Definition [Lottery Ticket Hypothesis]: Given a NN f with a parameter vector θ and an optimization function $O^N(f,\theta) = \theta^N$, which gives the optimized parameters of f after N training steps, there exists a sparse sub-network characterized by the binary mask M such that for some iteration K, $O^N(f,\theta^K * M)$ performs as well as $O^N(f,\theta) * M$, whereas the model trained from another random initialization θ_S, using the same mask $O^N(f,\theta_S * M)$, typically does not*. Frankle et al. (2019a) initially claimed the LTH held for $K=0$, but later revised this to $N \gg K \geq 0$ (Frankle et al., 2019b; Liu et al., 2019).

LTs enjoy significantly faster convergence compared to regular NN training but require the connectivity mask as found by the pruning solution (Frankle et al., 2019a) along with values from early training (Frankle et al., 2019b). Given the importance of the early phase of training (Frankle et al., 2020a; Lewkowycz et al., n.d.), it is natural to ask about the difference between lottery tickets and the solution they are derived from. Answering this question can help us understand if the success of LTs is primarily due to its relation to the solution, or if we can identify generalizable characteristics that help with sparse NNs training.

4 Experiments

Here we show empirically that (1) sparsity-aware initialization improves gradient flow at initialization for all methods, and achieves higher generalization for networks without BatchNorm, (2) the

*See Frankle et al. (2019b) for details. $*$ indicates element-wise multiplication, respecting the mask.

Table 1: **Results of Trained Sparse/Dense Models from Different Initializations.** The initializations proposed in Eq. (1) (Ours) and Liu et al. (2019) improve generalization consistently over masked dense (Original) except for in ResNet50. Note that VGG16 trained without a sparsity-aware initialization fails to converge in some instances. *Baseline* corresponds to the original dense architecture, whereas *Small Dense* corresponds to a smaller dense model with approximately the same parameter count as the sparse models.

	MNIST			ImageNet-2012					
	LeNet5 (95% sparse)			VGG16 (80% sparse)			ResNet50 (80% sparse)		
Baseline	99.20±0.05			69.25±0.13			76.75±0.12		
Small Dense	98.18±0.13			61.75±0.09			71.95±0.24		
	Original	Liu et al.	Ours	Original	Liu et al.	Ours	Original	Liu et al.	Ours
Scratch	94.40±3.41	94.86±3.41	**97.20**±0.18	51.81±3.02	**62.71**±0.05	62.52±0.10	70.58±0.18	**70.72**±0.16	70.63±0.22
SET	94.18±3.30	**96.35**±1.53	95.51±4.01	53.55±1.03	**63.19**±0.26	63.13±0.15	**72.93**±0.27	72.77±0.27	72.56±0.14
RigL	94.18±3.68	**97.76**±0.18	**97.76**±0.13	37.15±26.20	**63.69**±0.02	63.56±0.06	**74.41**±0.05	74.38±0.10	74.38±0.01

mask updates of DST methods increase gradient flow and create new negative eigenvalues in the Hessian; which we believe to be the main factor for improved generalization, (3) lottery tickets have poor gradient flow, however they achieve good performance by effectively re-learning the pruning solution, meaning they do not address the problem of training sparse NNs in general. Our experiments include the following settings: LeNet5 on MNIST, VGG16 on ImageNet-2012 and ResNet-50 on ImageNet-2012. Experimental details can be found in Appendix B[†].

4.1 Gradient Flow at Initialization

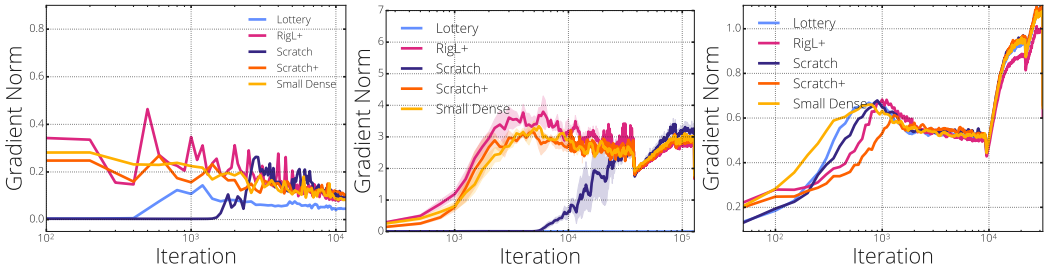
In this section, we measure the gradient flow over the course of the training (Fig. 2) and evaluate the performance of our generalized He initialization method (Table 1), and that proposed by Liu et al. (2019), over the commonly used masked dense initialization. Sparse NN initialized using the initialization distribution of a dense model (*Scratch* in Fig. 2) start in a flat region where gradient flow is very small and don't make any early progress. Learning starts after 1000 iterations for LeNet5 and 5000 for VGG-16, however, their generalization is sub-optimal. Liu et al. (2019) claim their proposed initialization has no empirical effect as compared to the masked dense initialization[‡]. Although technically incorrect (see §3.1), our results show their method to be largely as effective as our proposed initialization. This indicates that the assumption of a mask having roughly uniform mask sparsity is sufficient for the masks we considered. Both of these initializations remedy the vanishing gradient problem at initialization (*Scratch+* in Fig. 2) and result in better generalization for all methods. For instance, improved initialization results in an 11% improvement in Top-1 accuracy for VGG16 (62.52 vs 51.81). While initialization is extremely important for NNs without BatchNorm and skip connections, its effect on modern architectures, such as Resnet-50, is limited (Evci et al., 2019; Frankle et al., 2020c; Zhang et al., 2019). We confirm these observations in our ResNet-50 experiments in which, despite some initial improvement in gradient flow, our initialization seems to have no effect on final generalization. We observe significant increases in gradient norm after each learning rate drop (due to increased variance in gradients), which suggests studying gradient norm in the later part of the training might not be helpful. On the other hand, we observe a significant difference in gradient flow during training between sparse networks and small dense models of a similar parameter count. Can the performance gap between static-sparse and dense models be explained by this difference?

4.2 Gradient Flow during Training and Dynamic Sparse Training

In Fig. 2 we observed improved gradient flow for RigL. In this section we focus on those iterations in which the sparse connectivity is updated, and measure the change in gradient flow along with

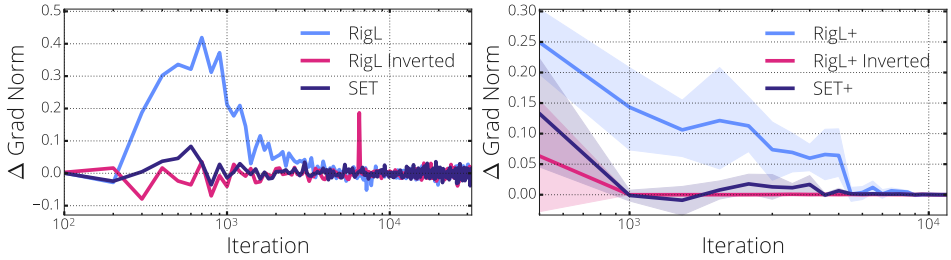
[†]Implementation of our sparse initialization, Hessian calculation and code for reproducing our experiments will be open sourced with the final version. Additionally we provide videos that shows the evolution of Hessian during training under different algorithms in the supplementary material.

[‡]Models with BatchNorm and skip connections are less affected by initialization, and this is likely why the authors did not observe this effect.



(a) LeNet5 (b) VGG-16 (c) ResNet-50

Figure 2: **Gradient Flow of Sparse Models during Training.** Gradient flow during training averaged over multiple runs, '+' indicates training runs with our proposed sparse initialization and Small Dense corresponds to training of a dense network with same number of parameters as the sparse networks.



(a) ResNet-50 (b) LeLeNet5

Figure 3: **Effect of Mask Updates in Dynamic Sparse Training.** Effect of mask updates on the gradient norm. *RigL Inverted* chooses connections with least magnitude. We measure the gradient norm before and after the mask updates and plot the Δ. '+' indicates proposed initialization and used in MNIST experiments.

the Hessian spectrum. We also run the inverted baseline for RigL (*RigL Inverted*), in which the growing criteria is reversed and connections with least gradient magnitudes are activated.

DST methods such as RigL replace low saliency connections during training. Assuming the pruned connections indeed have a low impact on the loss, we might expect to see increased gradient norm after new connections are activated, especially in the case of RigL, which picks new connections with high magnitude gradients. In Fig. 3 we confirm that RigL updates increase the norm of the gradient significantly, especially in the first half of training, whereas SET, which picks new connections randomly, seems to be less effective at this. Using the inverted RigL criteria doesn't improve the gradient flow, as expected, and without this RigL's performance degrades (73.83±0.12 for ResNet-50 and 92.71±7.67 for LeNet5). These results suggest that improving gradient flow early in training might be the key for training sparse networks and that is what RigL appears to be doing.

When the gradient is zero, or uninformative due to the error term of the approximation, analyzing the Hessian could provide additional insights (Ghorbani et al., 2019; Papyan, 2019; Sagun et al., 2017). In Appendix C, we show the Hessian spectrum before and after sparse connectivity updates. After RigL updates we observe more negative eigenvalues with significantly larger magnitudes as compared to SET. We leave investigating the relationship between gradient flow and the Hessian further as a future work.

4.3 Why Lotttery Tickets are Successful

We found that LTs do not improve gradient flow, either at initialization, or early in training, as shown in Fig. 2. This may be surprising given the apparent success of LTs, however the questions posed in §3.3 present an alternative hypothesis for the ease of training from a LT initialization. Here we present results showing that indeed (1) LTs initializations are consistently closer to the

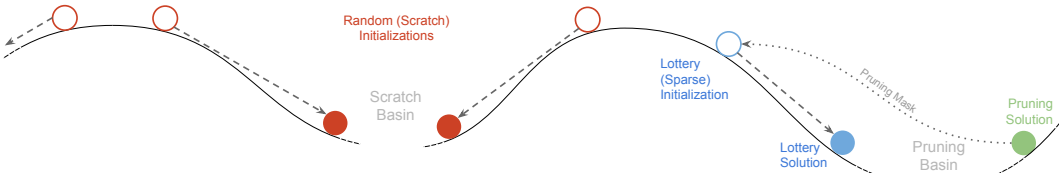
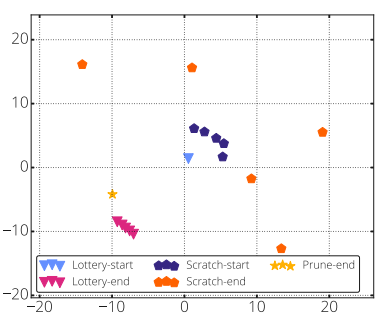


Figure 4: **Lottery Tickets Are Biased Towards the Pruning Solution, Unlike Random Initialization.** A cartoon illustration of the loss landscape of a sparse model, after it is pruned from a dense solution to create a LT sub-network. A lottery ticket initialization is within the basin of attraction of the pruned model's solution. In contrast a random initialization is unlikely to be close to the dense solution's basin.

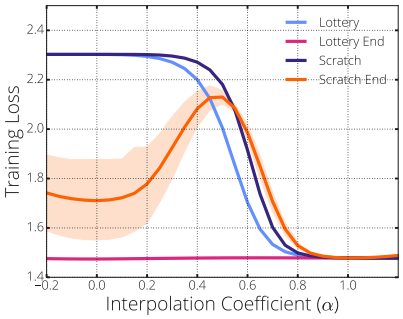


(a) MDS Projection

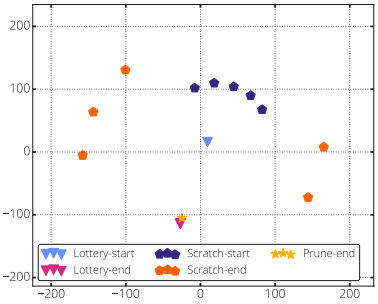
MNIST/LeNet5

	LT	Scratch
Acc_{test}	98.36	90.36
d_{start}	13.19	22.77
d_{end}	7.15	16.87

(b) L2 Distances



(c) Loss Interpolation

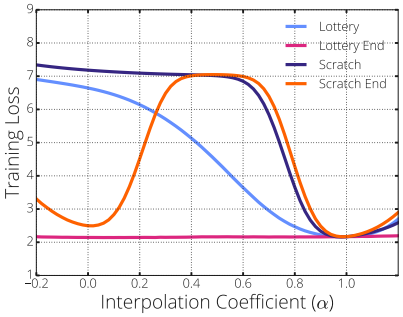


(d) MDS Projection

ImageNet-2012 /ResNet-50

Method	LT	Scratch
Acc_{test}	75.74	71.16
d_{start}	147.12	200.44
d_{end}	39.35	215.98

(e) L2 Distances



(f) Loss Interpolation

Figure 5: **MDS Embeddings/L2 Distances**: (a, d): 2D Multi-dimensional Scaling (MDS) embedding of sparse NNs with the same connectivity/mask; (b, e): the average L2-distance between a pruning solution and other derived sparse networks; (c, f): linear path between the pruning solution ($\alpha=1.0$) and LT/scratch at both initialization, and solution (end of training). Top and bottom rows are for MNIST/LeNet5 and ImageNet-2012/ResNet-50 respectively.

pruning solution than a random initialization, (2) trained LTs (i.e. LT solutions) consistently end up in the same basin as the pruning solution and (3), LT solutions are highly similar to pruning solutions under various function similarity measures. Our resulting understanding of LTs in the context of the pruning solution and the loss landscape is illustrated in Fig. 4.

Experimental Setup To investigate the relationship between the pruned and LT solutions we perform experiments on two models/datasets: a 95% sparse LeNet5[§] architecture (LeCun et al., 1989) trained on MNIST (where the original LT formulation works, i.e. $K=0$), and an 80% sparse ResNet-50 (Wu et al., 2018) on ImageNet-2012 (Russakovsky et al., 2015) (where $K=0$ doesn't work (Frankle et al., 2019b)), for which we use values from $K=2000$ ($\approx 6^{\text{th}}$ epoch). In both cases, we find a LT initialization by pruning each layer of a dense NN separately using

[§]Note: We use ReLU activation functions, unlike the original architecture (LeCun et al., 1989).

Table 2: **Ensemble & Prediction Disagreement**. We compare the function similarity (Fort et al., 2020) and ensemble generalization over 5 sparse models, trained from random initializations and LTs, with the original pruning solution. As a baseline, we also show results for 5 pruned models trained from different random initializations. See Appendix D for the complete results.

	Initialization	(Top-1) Test Acc.	Ensemble	Disagree.	Disagree. w/ Pruned
LeNet5 MNIST	LT	98.36±0.05	98.39	0.0057±0.0005	0.0096±0.0004
	Scratch	90.36±9.79	96.30	0.1436±0.1128	0.0930±0.0974
	Pruned Soln.	98.28	–	–	–
	5 Diff. Pruned	98.06±0.20	98.72	0.0222±0.0019	0.0209±0.0015*
ResNet50 ImageNet	LT	75.73±0.08	76.27	0.0894±0.0009	0.0941±0.0009
	Scratch	71.16±0.13	74.05	0.2039±0.0013	0.2033±0.0012
	Pruned Soln.	75.60	–	–	–
	5 Diff. Pruned	75.65±0.13	77.80	0.1620±0.0008	0.1623±0.0011*

* Here we compare 4 different pruned models with the pruning solution LT/Scratch are derived from.

magnitude-based iterative pruning (Zhu et al., 2018). Further details about our experiments can be found in Appendix B.

Lottery Tickets Are Close to the Pruning Solution We train 5 different models using different seeds from both *scratch* (random) and LT initializations, the results of which are in Figs. 5b and 5e. These networks share the same pruning mask and therefore lie in the same solution space. We visualize distances between initial and final points of these experiments in Figs. 5a and 5d using 2D Multi-dimensional Scaling (MDS) (Kruskal, 1964) embeddings. ***LeNet5/MNIST***: In Fig. 5b, we provide the average L2 distance to the pruning solution at initialization (d_{start}), and after training (d_{end}). We observe that LT initializations start significantly closer to the pruning solution on average ($d_{start}=13.19$ v.s. 22.77). After training, LTs end up more than $2\times$ closer to the pruning solution compared to scratch. ***Resnet-50/ImageNet-2012:*** We observe similar results for Resnet-50/ImageNet-2012. LTs, again, start closer to the pruning solution, and solutions are $5\times$ closer ($d_{end}=39.35$ v.s. 215.98). With these observations, non-random initial loss values for LT initialization reported first by (Zhou et al., 2019) seem reasonable. LTs are biased towards the pruning solution they are derived from, but are they in the same basin?

Lottery Tickets are in the Pruning Solution Basin Investigating paths between different solutions is a popular tool for understanding how various points in parameter space relate to each other in the loss landscape (Draxler et al., 2018; Evci et al., 2019; Fort et al., 2020; Frankle et al., 2020b; Garipov et al., 2018; Goodfellow et al., 2015). For example, Frankle et al. (2019b) use linear interpolations to show that LTs always go to the same basin when trained in different data orders. In Figs. 5c and 5f we look at the linear paths between pruning solution and 4 other points: LT initialization/solution and random (scratch) initialization/solution. Each experiment is repeated 5 times with different random seeds, and mean values are provided with 80% confidence intervals. In both experiments we observe that the linear path between LT initialization and the pruning solution decreases faster compared to the path that originates from scratch initialization. After training, the linear paths towards the pruning solution change drastically. The path from the scratch solution depicts a loss barrier; the scratch solution seems to be in a different basin than the pruning solution[¶]. In contrast, LTs are linearly connected to the pruning solution in both small and large-scale experiments indicating that LTs have the same basin of attraction as the pruning solutions they are derived from. While it seems likely, these results do not however explicitly show that the LT and pruning solutions have learned similar functions.

Lottery Tickets Learn Similar Functions to the Pruning Solution Fort et al. (2020) motivate deep ensembles by empirically showing that models starting from different random initializations typically learn different solutions, as compared to models trained from similar ini-

[¶]This is not always true, it is possible that non-linear low energy paths exist between two solutions (Draxler et al., 2018; Garipov et al., 2018), but searching for such paths is outside the scope of this work.

tializations. Here we adopt the analysis of (Fort et al., 2020), but for comparing LT initializations and random initializations. In Table 2 we show the mean fractional disagreement over all pairs of models, i.e. the fraction of class predictions for which pairs of models disagree. The fractional disagreement with the pruning solution is the fraction of class predictions over which the LT and scratch models disagree with the pruning solution they were derived from.

The results presented in Table 2 suggest that all 5 LTs models converge on a solution almost identical to the pruning solution. Interestingly, the 5 LT models are even more similar to each other than the pruning solution, possibly because they share an initialization and training is stable (Frankle et al., 2019b). On the other hand, scratch solutions are more diverse as they start from different initializations. As suggested by the analysis of Fort et al. (2020), ensembles of different solutions are more robust, and generalize better, than ensembles of similar solutions. An ensemble of 5 LT models with low disagreement doesn't significantly improve generalization as compared to an ensemble of 5 different pruning solutions with similar individual test accuracy. We further demonstrate these results by comparing the output probability distributions using the Kullback–Leibler Divergence (KL), and Jensen–Shannon Divergence (JSD) in Appendix D.

Implications: **(a) Rewinding of LTs.** Frankle et al. (2019b, 2020b) argued that LTs work when the training is *stable*, and thus converges to the same basin when trained with different data sampling orders. In §4.3, we show that this basin is the same one found by pruning, and since the training converges to the same basin as before, we expect to see limited gains from rewinding if any. This is partially confirmed by Renda et al. (2020) which shows that restarting the learning rate schedule from the pruning solution performs better than rewinding the weights. **(b) Transfer of LTs.** Given the close relationship between LTs and pruning solutions, the observation that LTs trained on large datasets transfer to smaller ones, but not *vice versa* (Morcos et al., 2019; Sabatelli et al., 2020) can be explained by a common observation in transfer learning: networks trained in large datasets transfer to smaller ones. **(c) LT's Robustness to Perturbations.** Frankle et al. (2020a) and Zhou et al. (2019) found that certain perturbations, like only using the signs of weights at initialization, do not impact LT generalization, while others, like shuffling the weights, do. Our results bring further insights to these observations: As long as the perturbation is small enough such that a LT stays in the same basin of attraction, results will be as good as the pruning solution. **(d) Success of LTs.** While it is exciting to see widespread applicability of LTs in different domains (Brix et al., 2020; Li et al., 2020; Venkatesh et al., 2020), the results presented in this paper suggest this success may be due to the underlying pruning algorithm (and transfer learning) rather than LT initializations themselves.

5 Conclusion

We attempted to answer the questions of (1) why training unstructured sparse networks from random initialization performs poorly and; (2) what makes Lottery Tickets (LTs) and Dynamic Sparse Training (DST) the exceptions? We identified that randomly initialized unstructured sparse Neural Networks (NNs) exhibit poor gradient flow when initialized naively and proposed an alternative initialization that scales the initial variance for each neuron separately. Furthermore we showed that modern sparse NN architectures are more sensitive to poor gradient flow during early training rather than initialization alone. We observed that this is somewhat addressed by state-of-the-art DST methods, such as Rigged Lottery (RigL), which significantly improves gradient flow during early training over traditional sparse training methods. Finally, we show that LTs do not improve gradient flow at either initialization or during training, but rather their success lies in effectively re-learning the original pruning solution they are derived from. We showed that a LTs initialization resides within the same basin of attraction as the pruning solution and, furthermore, when trained the LT solution learns a highly similar solution to the pruning solution. These findings suggest that LTs are fundamentally limited in their potential for improving the training of sparse NNs more generally.

References

Brix, Christopher, Parnia Bahar, and Hermann Ney (2020). "Successfully Applying the Stabilized Lottery Ticket Hypothesis to the Transformer Architecture". In: *ArXiv*. URL: https://arxiv.org/abs/2005.03454.

Castellano, Giovanna, Anna Maria Fanelli, and Marcello Pelillo (1997). "An iterative pruning algorithm for feedforward neural networks". In: *IEEE Transactions on Neural Networks*. ISSN: 1045-9227. DOI: 10.1109/72.572092.

Dettmers, Tim and Luke Zettlemoyer (2019). "Sparse Networks from Scratch: Faster Training without Losing Performance". In: *ArXiv*. URL: http://arxiv.org/abs/1907.04840.

Draxler, Felix et al. (2018). "Essentially No Barriers in Neural Network Energy Landscape". In: *International Conference on Machine Learning*.

Evci, Utku et al. (2019). "The Difficulty of Training Sparse Neural Networks". In: *ArXiv*. URL: http://arxiv.org/abs/1906.10732.

Evci, Utku et al. (2020). "Rigging the Lottery: Making All Tickets Winners". In: *Proceedings of Machine Learning and Systems 2020*.

Fort, Stanislav, Huiyi Hu, and Balaji Lakshminarayanan (2020). "Deep Ensembles: A Loss Landscape Perspective". In: *International Conference on Learning Representations*.

Frankle, Jonathan and Michael Carbin (2019a). "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *7th International Conference on Learning Representations (ICLR)*.

Frankle, Jonathan, David J. Schwab, and Ari S. Morcos (2020a). "The Early Phase of Neural Network Training". In: *International Conference on Learning Representations*.

Frankle, Jonathan et al. (2019b). "Stabilizing the Lottery Ticket Hypothesis". In: *ArXiv*. URL: https://arxiv.org/abs/1903.01611.

— (2020b). "Linear Mode Connectivity and the Lottery Ticket Hypothesis". In: *Proceedings of the International Conference on Machine Learning*.

— (2020c). "Pruning Neural Networks at Initialization: Why are We Missing the Mark?" In: *ArXiv*. URL: https://arxiv.org/abs/2009.08576.

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

Garipov, Timur et al. (2018). "Loss Surfaces, M Connectivity, and Fast Ensembling of DNNs". In: *Advances in Neural Information Processing Systems*.

Ghorbani, Behrooz, Shankar Krishnan, and Ying Xiao (2019). "An Investigation into Neural Net Optimization via Hessian Eigenvalue Density". In: *Proceedings of the 36th International Conference on Machine Learning*. URL: http://proceedings.mlr.press/v97/ghorbani19b.html.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Goodfellow, Ian J., Oriol Vinyals, and Andrew M. Saxe (2015). "Qualitatively characterizing neural network optimization problems". In: *International Conference on Learning Representations*.

Han, Song et al. (2015). "Learning both weights and connections for efficient neural network". In: *Advances in neural information processing systems*.

Han, Song et al. (2016). "EIE: Efficient Inference Engine on Compressed Deep Neural Network". In: *Proceedings of the 43rd International Symposium on Computer Architecture*.

Hanson, Stephen José and Lorien Y. Pratt (1988). "Comparing biases for minimal network construction with back-propagation". In: *Proceedings of the 1st International Conference on Neural Information Processing Systems (NIPS)*. Morgan Kaufmann.

Hassibi, B. and D. Stork (1993). "Second order derivatives for network pruning: Optimal Brain Surgeon". In: *Advances in Neural Information Processing Systems*.

He, Kaiming et al. (2015). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *IEEE International Conference on Computer Vision (ICCV)*.

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32Nd International*

Conference on International Conference on Machine Learning.

Kruskal, J. (1964). "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis". In: *Psychometrika*. DOI: 10.1007/BF02289565.

Kusupati, Aditya et al. (2020). "Soft Threshold Weight Reparameterization for Learnable Sparsity". In: *Proceedings of the International Conference on Machine Learning*.

LeCun, Yann, John S. Denker, and Sara A. Solla (1990). "Optimal Brain Damage". In: *Advances in Neural Information Processing Systems*.

LeCun, Yann et al. (1989). "Backpropagation applied to handwritten zip code recognition". In: *Neural Computation*. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541.

Lee, Namhoon, Thalaiyasingam Ajanthan, and Philip H. S. Torr (2019). "SNIP: Single-shot Network Pruning based on Connection Sensitivity". In: *International Conference on Learning Representations (ICLR), 2019*.

Lewkowycz, Aitor et al. (n.d.). "The large learning rate phase of deep learning: the catapult mechanism". In: *Arxiv* (). URL: https://arxiv.org/pdf/2003.02218.

Li, Bai et al. (2020). "Towards Practical Lottery Ticket Hypothesis for Adversarial Training". In: *ArXiv*. URL: https://arxiv.org/abs/2003.05733.

Liu, Zhuang et al. (2019). "Rethinking the Value of Network Pruning". In: *International Conference on Learning Representations*.

Louizos, Christos, Karen Ullrich, and Max Welling (2017). "Bayesian compression for deep learning". In: *Advances in Neural Information Processing Systems*.

Malach, Eran et al. (2020). "Proving the Lottery Ticket Hypothesis: Pruning is All You Need". In: *ArXiv*. URL: http://arxiv.org/abs/2002.00585.

Mocanu, Decebal Constantin et al. (2018). "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science". In: *Nature Communications*.

Molchanov, Dmitry, Arsenii Ashukha, and Dmitry Vetrov (2017). "Variational Dropout Sparsifies Deep Neural Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR. URL: http://proceedings.mlr.press/v70/molchanov17a.html.

Molchanov, Pavlo et al. (2016). "Pruning Convolutional Neural Networks for Resource Efficient Inference". In: *ArXiv*. URL: http://arxiv.org/abs/1611.06440.

Morcos, Ari et al. (2019). "One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers". In: *Advances in Neural Information Processing Systems 32*.

Mostafa, Hesham and Xin Wang (2019). "Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. URL: http://proceedings.mlr.press/v97/mostafa19a.html.

Mozer, Michael C. and Paul Smolensky (1989a). "Using Relevance to Reduce Network Size Automatically". In: *Connection Science*. DOI: 10.1080/09540098908915626.

— (1989b). "Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment". In: *Advances in Neural Information Processing Systems 1*.

Papyan, Vardan (2019). "Measurements of Three-Level Hierarchical Structure in the Outliers in the Spectrum of Deepnet Hessians". In: *ArXiv*. URL: https://arxiv.org/abs/1901.08244.

Ramanujan, Vivek et al. (2019). "What's Hidden in a Randomly Weighted Neural Network?" In: *ArXiv*. URL: http://arxiv.org/abs/1911.13299.

Renda, Alex, Jonathan Frankle, and Michael Carbin (2020). "Comparing Rewinding and Fine-tuning in Neural Network Pruning". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=S1gSj0NKvB.

Russakovsky, Olga et al. (2015). "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)*.

Sabatelli, Matthia, Mike Kestemont, and Pierre Geurts (2020). "On the Transferability of Winning Tickets in Non-Natural Image Datasets". In: *ArXiv*. URL: https://arxiv.org/abs/2005.05232.

Sagun, Levent et al. (2017). "Empirical Analysis of the Hessian of Over-Parametrized Neural Networks". In: *International Conference on Learning Representations*.

Setiono, Rudy (1997). "A Penalty-Function Approach for Pruning Feedforward Neural Networks". In: *Neural Computation*. DOI: 10.1162/neco.1997.9.1.185.

Sietsma, Jocelyn and Robert J.F. Dow (1988). "Neural net pruning-why and how". In: *IEEE International Conference on Neural Networks*. DOI: 10.1109/ICNN.1988.23864.

Tanaka, Hidenori et al. (2020). "Pruning neural networks without any data by iteratively conserving synaptic flow". In: *ArXiv*. URL: https://arxiv.org/abs/2006.05467.

van der Walt, S., S. C. Colbert, and G. Varoquaux (2011). "The NumPy Array: A Structure for Efficient Numerical Computation". In: *Computing in Science Engineering*.

Venkatesh, Bindya et al. (2020). "Calibrate and Prune: Improving Reliability of Lottery Tickets Through Prediction Calibration". In: *ArXiv*. URL: http://arxiv.org/abs/2002.03875.

Wang, Chaoqi, Guodong Zhang, and Roger Grosse (2020). "Picking Winning Tickets Before Training by Preserving Gradient Flow". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=SkgsACVKPH.

Wortsman, Mitchell, Ali Farhadi, and Mohammad Rastegari (2019). "Discovering Neural Wirings". In: *Advances in Neural Information Processing Systems*.

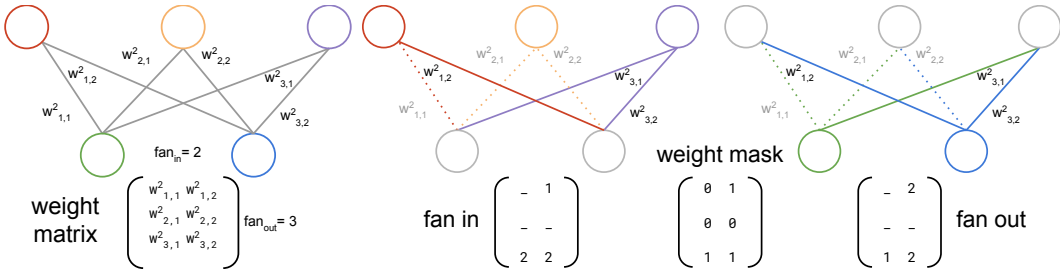
Wu, Songtao, Shenghua Zhong, and Yan Liu (2018). "Deep residual learning for image steganalysis". In: *Multimedia Tools and Applications*.

Xiao, Lechao et al. (2018). "Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks". In: *International Conference on Machine Learning (ICML)*.

Zhang, Hongyi, Yann N. Dauphin, and Tengyu Ma (2019). "Fixup Initialization: Residual Learning Without Normalization". In: *International Conference on Learning Representations*.

Zhou, Hattie et al. (2019). "Deconstructing lottery tickets: Zeros, signs, and the supermask". In: *Advances in Neural Information Processing Systems*.

Zhu, Michael and Suyog Gupta (2018). "To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression". In: *International Conference on Learning Representations Workshop*.



(a) Dense Layer (b) Sparse Layer

Figure 6: **Glorot/He Initialization for a Sparse NN.** (Glorot et al., 2010; He et al., 2015) restrict the outputs of all neurons to be zero-mean and of unit variance. All neurons in a dense NN layer (a) have the same fan-in/fan-out, whereas in a sparse NN (b) the fan-in/fan-out can differ for *every neuron*, potentially requiring sampling from a different distribution for every neuron. The fan-in matrix contains the values used in Eq. (1) for each neuron.

A Glorot/He Initialization Generalized to Neural Networks with Heterogeneous Connectivity: Full Explanation/Derivation

Here we derive the full generalized initialization for both the forwards/backwards cases (i.e. fan-in/fan-out), refer to Fig. 6 for an illustration of how the connectivity for the fan-in/fan-out cases are determined for each neuron.

A.1 Generalized Glorot/He Initialization: Backwards, Forwards and Average Cases

For every weight $w_{ij}^{[\ell]} \in W^{n^{[\ell]} \times n^{[\ell-1]}}$ in a layer ℓ with $n^{[\ell]}$ neurons, connecting neuron i in layer ℓ to neuron j in layer $(\ell-1)$ with $n^{[\ell-1]}$ neurons, and weight mask $[m_{ij}^{[\ell]}] = M^{\ell} \in [0,1]^{n^{[\ell]} \times n^{[\ell-1]}}$,

$$\begin{aligned} &\text{Glorot et al. (2010): } w_{ij}^{[\ell]} \sim \mathcal{N}\left(0, \tfrac{1}{u}\right) \\ &\text{He et al. (2015):} \qquad w_{ij}^{[\ell]} \sim \mathcal{N}\left(0, \tfrac{2}{u}\right) \end{aligned} \quad \text{where } u = \begin{cases} \textit{fan-in}_i^{[\ell]} & \text{(forward)} \\ \textit{fan-out}_j^{[\ell]} & \text{(backward)} \\ \left(\textit{fan-in}_i^{[\ell]} + \textit{fan-out}_j^{[\ell]}\right)/2 & \text{(average)} \end{cases} \tag{2}$$

where,

$$\textit{fan-in}_i^{[\ell]} = \sum_{j=1}^{n^{[\ell-1]}} m_{ij}^{[\ell]}, \qquad \textit{fan-out}_j^{[\ell]} = \sum_{i=1}^{n^{[\ell]}} m_{ij}^{[\ell]},$$

are the number of incoming and outgoing connections respectively. In the special case of a dense layer where $m_{ij}^{[\ell]} = 1, \forall i,j$, Eq. (1) reduces to the initializations proposed by (Glorot et al., 2010; He et al., 2015) since $\textit{fan-in}_i^{[\ell]} = n^{[\ell-1]}, \forall i$, and $\textit{fan-out}_j^{[\ell]} = n^{[\ell]}, \forall j$.

A.2 Derivation: Fixed Mask, Forward Propagation

Given a sparse NN, where the output of a neuron a_i is given by, $a_i^{[\ell]} = f\left(z_i^{[\ell]}\right)$, where $z_i^{[\ell]} = \sum_j^{n^{[\ell-1]}} m_{ij}^{[\ell]} w_{ij}^{[\ell]} a_j^{[\ell-1]}$, where $m_{ij}^{[\ell]} \in M^{[\ell]}$ and $w_{ij}^{[\ell]} \in W^{[\ell]}$ are the mask and weights respectively for layer ℓ, and $a_j^{[\ell-1]}$ the output of the previous layer. Assume the mask $M^{[\ell]} \in \mathbb{1}^{n^{[\ell]} \times n^{[\ell-1]}}$ is constant, where $\mathbb{1}^{n^{[\ell]} \times n^{[\ell-1]}}$ is an indicator matrix.

As in Glorot et al. (2010) we want to ensure $\text{Var}(a_i^{[\ell]}) = \text{Var}(a_i^{[\ell-1]})$, and $\text{mean}(a_i^{[\ell]}) = 0$. Assume that $f(x) \approx x$ for x close to 0, e.g. in the case of $f(x) = \tanh(x)$, and that $w_{ij}^{[\ell]}$ and $a_j^{[\ell-1]}$

are independent,

$$\begin{aligned}
\mathrm{Var}(a_i^{[\ell-1]}) &\approx \mathrm{Var}(z_i^{[\ell]}) && (3)\\
&= \mathrm{Var}\left(\sum_{j=1}^{n^{[l-1]}} m_{ij}^{[\ell]} w_{ij}^{[\ell]} a_j^{[\ell-1]}\right) && (4)\\
&= \sum_{j=1}^{n^{[\ell-1]}} \mathrm{Var}\left(m_{ij}^{[\ell]} w_{ij}^{[\ell]} a_j^{[\ell-1]}\right) && \text{(independent sum)} \quad (5)\\
&= \sum_{j=1}^{n^{[\ell-1]}} \left(m_{ij}^{[\ell]}\right)^2 \mathrm{Var}\left(w_{ij}^{[\ell]} a_j^{[\ell-1]}\right) && \because m_{ij}^{[\ell]} \text{ is constant}, \mathrm{Var}(cX) = c^2\mathrm{Var}(X) \quad (6)\\
&= \sum_{j=1}^{n^{[\ell-1]}} m_{ij}^{[\ell]} \mathrm{Var}\left(w_{ij}^{[\ell]} a_j^{[\ell-1]}\right) && \because m_{ij}^{[\ell]} \in [0,1], \left(m_{ij}^{[\ell]}\right)^2 = m_{ij}^{[\ell]}. \quad (7)\\
&= \sum_{j=1}^{n^{[\ell-1]}} m_{ij}^{[\ell]} \mathrm{Var}(w_{ij}^{[\ell]}) \mathrm{Var}(a_j^{[\ell-1]}). && \text{(independent product)} \quad (8)
\end{aligned}$$

Assume $\mathrm{Var}(w_{im}^{[\ell]}) = \mathrm{Var}(w_{in}^{[\ell]}), \forall n,m$, i.e. the variance of all weights for a given neuron are the same, and $\mathrm{Var}(a_n^{[\ell-1]}) = \mathrm{Var}(a_m^{[\ell-1]})$, i.e. the variance of any of the outputs of the previous layer are the same. Therefore we can simplify Eq. (8),

$$\begin{aligned}
\mathrm{Var}(a_i^{[\ell-1]}) &= \sum_{j=1}^{n^{[\ell-1]}} m_{ij}^{[\ell]} \mathrm{Var}(w_{ij}^{[\ell]}) \mathrm{Var}(a_j^{[\ell-1]}) && (9)\\
&= \mathrm{Var}(w_{ij}^{[\ell]}) \mathrm{Var}(a_j^{[\ell-1]}) \sum_{j=1}^{n^{[\ell-1]}} m_{ij}^{[\ell]}. && (10)
\end{aligned}$$

Let neuron i's number of non-masked weights be denoted $\textit{fan-in}_i^{[\ell]}$, where $\textit{fan-in}_i^{[\ell]} = \sum_{j=1}^{n^{[\ell-1]}} m_{ij}^{[\ell]}$, then

$$\begin{aligned}
\mathrm{Var}(a_i^{[\ell-1]}) &= \textit{fan-in}_i^{[\ell]} \mathrm{Var}(w_{ij}^{[\ell]}) \mathrm{Var}(a_j^{[\ell-1]}) && (11)\\
\text{Recall, } \mathrm{Var}(a_i^{[\ell-1]}) &= \mathrm{Var}(a_j^{[\ell-1]})\\
\Rightarrow \mathrm{Var}(w_{ij}^{[\ell]}) &= \frac{1}{\textit{fan-in}_i^{[\ell]}}. && (12)
\end{aligned}$$

Therefore, in order to have the output of each neuron $a_i^{[\ell]}$ in layer ℓ to have unit variance, and mean 0, we need to sample the weights for each neuron from the normal distribution,

$$[w_{ij}^{[\ell]}] \sim \mathcal{N}\left(0, \frac{1}{\textit{fan-in}_i^{[\ell]}}\right), \tag{13}$$

where s_i^ℓ is the sparsity of weights of the neuron with output a_i. For the ReLU activation function, following the derivation in He et al. (2015),

$$[w_{ij}^{[\ell]}] \sim \mathcal{N}\left(0, \frac{2}{\textit{fan-in}_i^{[\ell]}}\right). \tag{14}$$

A.3 Fixed Mask: Backward Pass

Given a sparse NN, where the output of a neuron a_i is given by, $a_i^{[\ell]} = f\left(z_i^{[\ell]}\right)$, where $z_i^{[\ell]} = \sum_j^{n^{[\ell-1]}} m_{ij}^{[\ell]} w_{ij}^{[\ell]} a_j^{[\ell-1]}$, where $m_{ij}^{[\ell]} \in M^{[\ell]}$ and $w_{ij}^{[\ell]} \in W^{[\ell]}$ are the mask and weights respectively

for layer ℓ, and $a_j^{[\ell-1]}$ the output of the previous layer. Assume the mask $M^{[\ell]} \in \mathbb{1}^{n^{[\ell]} \times n^{[\ell-1]}}$ is constant, where $\mathbb{1}^{n^{[\ell]} \times n^{[\ell-1]}}$ is an indicator matrix, and let $L\big(\theta = \{W^{[\ell]}, \ell = 0...N\}\big)$ be the loss we are optimizing.

As in Glorot et al. (2010), from the backward-propagation standpoint, we want to ensure $\mathrm{Var}(\frac{\partial L}{\partial z_i^{[\ell]}}) = \mathrm{Var}(\frac{\partial L}{\partial z_i^{[\ell-1]}})$, and $\mathrm{mean}(\frac{\partial L}{\partial z_i^{[\ell]}}) = 0$. Assume that $f'(0) = 1$,

$$\mathrm{Var}(\frac{\partial L}{\partial z_j^{[\ell]}}) \approx \mathrm{Var}(\frac{\partial L}{\partial a_j^{[\ell-1]}}) \tag{15}$$

$$= \mathrm{Var}\left(\sum_{i=1}^{n^{[\ell]}} m_{ij}^{[\ell]} w_{ij}^{[\ell]} \frac{\partial L}{\partial z_i^{[\ell]}}\right) \tag{16}$$

$$= \sum_{i=1}^{n^{[\ell]}} \mathrm{Var}\left(m_{ij}^{[\ell]} w_{ij}^{[\ell]} \frac{\partial L}{\partial z_i^{[\ell]}}\right) \qquad \text{(independent sum)} \tag{17}$$

$$= \sum_{i=1}^{n^{[\ell]}} \left(m_{ij}^{[\ell]}\right)^2 \mathrm{Var}\left(w_{ij}^{[\ell]} \frac{\partial L}{\partial z_i^{[\ell]}}\right) \qquad \because m_{ij}^{[\ell]} \text{ is constant}, \mathrm{Var}(cX) = c^2 \mathrm{Var}(X) \tag{18}$$

$$= \sum_{i=1}^{n^{[\ell]}} m_{ij}^{[\ell]} \mathrm{Var}\left(w_{ij}^{[\ell]} \frac{\partial L}{\partial z_i^{[\ell]}}\right) \qquad \because m_{ij}^{[\ell]} \in [0,1], \left(m_{ij}^{[\ell]}\right)^2 = m_{ij}^{[\ell]}. \tag{19}$$

$$= \sum_{i=1}^{n^{[\ell]}} m_{ij}^{[\ell]} \mathrm{Var}(w_{ij}^{[\ell]}) \mathrm{Var}(\frac{\partial L}{\partial z_i^{[\ell]}}). \qquad \text{(independent product)} \tag{20}$$

Assume $\mathrm{Var}(w_{mj}^{[\ell]}) = \mathrm{Var}(w_{nj}^{[\ell]}), \forall n,m$, i.e. the variance of all weights for a given neuron are the same, and $\mathrm{Var}(\frac{\partial L}{\partial z_n^{[\ell]}}) = \mathrm{Var}(\frac{\partial L}{\partial z_m^{[\ell]}})$, i.e. the variance of the output gradients of each neuron at layer l are the same. Then we can simplify Eq. (20),

$$\mathrm{Var}(\frac{\partial L}{\partial z_j^{[\ell]}}) = \sum_{i=1}^{n^{[\ell]}} m_{ij}^{[\ell]} \mathrm{Var}(w_{ij}^{[\ell]}) \mathrm{Var}(\frac{\partial L}{\partial z_i^{[\ell]}}) \tag{21}$$

$$= \mathrm{Var}(w_{ij}^{[\ell]}) \mathrm{Var}(\frac{\partial L}{\partial z_i^{[\ell]}}) \sum_{i=1}^{n^{[\ell]}} m_{ij}^{[\ell]}. \tag{22}$$

Let neuron i's number of non-masked weights be denoted $\textit{fan-out}_j^{[\ell]}$, where $\textit{fan-out}_j^{[\ell]} = \sum_{i=1}^{n^{[\ell]}} m_{ij}^{[\ell]}$, then

$$\mathrm{Var}(\frac{\partial L}{\partial z_j^{[\ell]}}) = \textit{fan-out}_j^{[\ell]} \mathrm{Var}(w_{ij}^{[\ell]}) \mathrm{Var}(\frac{\partial L}{\partial z_i^{[\ell]}}) \tag{23}$$

$$\text{Recall, } \mathrm{Var}(\frac{\partial L}{\partial z_j^{[\ell]}}) = \mathrm{Var}(\frac{\partial L}{\partial z_i^{[\ell]}})$$

$$\Rightarrow \mathrm{Var}(w_{ij}^{[\ell]}) = \frac{1}{\textit{fan-out}_j^{[\ell]}}. \tag{24}$$

Therefore, in order to have the output of each neuron $a_i^{[\ell]}$ in layer ℓ to have unit variance, and mean 0, we need to sample the weights for each neuron from the normal distribution,

$$[w_{ij}^{[\ell]}] \sim \mathcal{N}\left(0, \frac{1}{\textit{fan-in}_i^{[\ell]}}\right), \tag{25}$$

Table 3: **§4.3: Experiment Details/Hyperparameters**. Initial Learning Rate (LR), LR Schedule (Sched.), Batchsize (Batch.), Momentum (m), Weight Decay (WD), t_{start}, t_{end} and f are the pruning starting iteration, end iteration, and mask update frequency respectively.

Dataset	Model	t_{total}	Epochs	Batch.	LR	Sched.	m	WD	Sparsity	Pruning		
										t_{start}	t_{end}	f
MNIST	LeNet5	11719	30	128	0.1	Cosine	0.9	0	95%	3000	7000	100
ImageNet	ResNet50	32000	≈102	4096	1.6	Step*	0.9	1×10^{-4}	80%	5000	8000	2000

* Step schedule has a linear warm-up in first 5 epochs and decreases the learning rate by a factor of 10 at epochs 30,70 and 90.

Table 4: **§4.1: Experiment Details/Hyperparameters**. Initial Learning Rate (LR), LR Schedule (Sched.), Batchsize (Batch.), Momentum (m), Weight Decay (WD), Initial Drop Fraction (Drop.), t_{end} and f are the pruning mask update frequency and end iteration respectively. *LeNet5+* row corresponds the LeNet5 experiments with our sparse initialization, whereas *LeNet5* is the regular masked initialization.

Dataset	Model	t_{total}	Epochs	Batch.	LR	Sched.	m	WD	Sparsity	DST		
										Drop.	f	t_{end}
MNIST	LeNet5+ / LeNet5	11719	30	128	0.1	Cosine	0.9	$\frac{2\times10^{-4}}{1\times10^{-5}}$	95%	0.3 / 0.001	500	11719
ImageNet	ResNet50	32000	≈102	4096	1.6	Step*	0.9	1×10^{-4}	80%	0.3	100	25000
	VGG16	128000		1024	0.04					0.1	500	

* Step schedule has a linear warm-up in first 5 epochs and decreases the learning rate by a factor of 10 at epochs 30,70 and 90.

where s_i^ℓ is the sparsity of weights of the neuron with output a_i. For the ReLU activation function, following the derivation in He et al. (2015),

$$[w_{ij}^{[\ell]}] \sim \mathcal{N}\left(0, \frac{2}{\textit{fan-in}_i^{[\ell]}}\right). \tag{26}$$

B Experimental Details

B.1 Details of Experiments in Section 4.3

The training hyper-parameters used in §4.3 are shared in Table 3. All experiments in this section start with a pruning experiment, after which the sparsity masks found by pruning are used to perform LT experiments. We use iterative magnitude pruning (Zhu et al., 2018) in our experiments, which is a well studied and more efficient pruning method as compared to the one used by Frankle et al. (2019a). Our pruning algorithm performs iterative pruning without rewinding the weights between intermediate steps and requires significantly less iterations. We expect our results would be even more pronounced with additional rewinding steps.

We use SGD with momentum in all of our experiments. *Scratch* and *Lottery* experiments use the same hyper-parameters. Additional specific details of our experiments are shared below.

LeNet5 We prune all layers of LeNet5, so that they reach 95% final sparsity (i.e. 95% of the parameters are zeros). We choose this sparsity, since at this sparsity, we start observing stark differences between *Lottery* and *Scratch* in terms of performance. We observed instability (Frankle et al., 2019b) when we use weight decay for finding LTs and therefore set the weight decay to zero, similar to the MNIST experiments done in the original LT paper (Frankle et al., 2019a). Loss values for the linear interpolation experiments are calculated on the entire training set.

ResNet50 We prune all layers of ResNet50, except the first layer, so that they reach 80% final sparsity. In this setting rewinding to the original initialization doesn't work, hence we use values

from 6^{th} epoch. Loss values for the linear interpolation experiments are calculated using 500,000 images from the ImageNet-2012 training set.

B.2 Details of Experiments in Section 4.1 and 4.2

Training hyper-parameters used for these experiments are shared in Table 4.

MNIST In this setting, the hyper-parameters are almost same as in §4.3, except we enable weight decay as it brings better generalization. We use the masks found by pruning experiments in all of our MNIST experiments in this section. Different seeds use different masks. We simplify the update schedule of Dynamic Sparse Training (DST) methods such that they decay with learning rate. This approach fits well, since the original decay function used in these experiments is the cosine decay which is the same as our learning rate schedule. We scale learning rate such that it matches the initial drop fraction provided. Mask update frequency and initial drop fraction are chosen from a grid search of{50, 100, 500} and{0.1,0.3,0.5} respectively.

Hessian calculation The Hessian is calculated on full training set using Hessian-vector products. We mask our network after each gradient call and calculate only non-zero rows. After calculating the full Hessian, we use `numpy.eigh` (van der Walt et al., 2011) to calculate eigenvalues of the Hessian.

ImageNet-2012 In this setting, hyper-parameters are almost the same as in §4.3 except for VGG16 architecture, where we use a smaller batch size and learning rate. For all DST methods, we use a cosine drop schedule Dettmers et al., 2019 and hyper-parameters proposed by Evci et al. (2019). For VGG, we reduce the mask update frequency and the initial drop fraction, as we observe better performance after doing a grid search over{50, 100, 500} and{0.1,0.3,0.5} respectively. We also use a non-uniform (ERK) sparsity distribution among layers as described in Evci et al. (2020), since we observed that it brings better performance.

C Hessian Spectrum of LeNet5

Given a loss function L and parameters θ, we can write the first order Taylor approximation of the change in loss $\Delta L = L(\theta^{t+1}) - L(\theta^t)$ after a single training step with the learning rate $\epsilon > 0$ as :

$$\Delta L \approx -\epsilon \nabla L(\theta)^T \nabla L(\theta). \tag{27}$$

Note that as long as the error is small, gradient descent is guaranteed to decrease the loss by an amount proportional to $\nabla L(\theta)^T \nabla L(\theta)$, which we refer as the *gradient flow*. In practice large learning rates are used, and the first order approximation might not be accurate. Instead we can look at the second order approximation of ΔL:

$$\Delta L \approx -\alpha \nabla L(\theta)^T \nabla L(\theta) + \frac{\alpha^2}{2} \nabla L(\theta)^T H(\theta) \nabla L(\theta), \tag{28}$$

where $H(\theta)$ is the Hessian of the loss function. The eigenvalue spectrum of Hessian can help us understand the local landscape (Sagun et al., 2017), and help us identify optimization difficulties (Ghorbani et al., 2019). For example, if and when the gradient is aligned with large magnitude eigenvalues, the second term of Eq. (28) can have a significant effect on the optimization of L. If the gradient is aligned with large positive eigenvalues, it can prevent gradient descent from decreasing the loss and harm the optimization. Similarly, if it is aligned with negative eigenvalues it can help to accelerate optimization.

We show the Hessian spectrum before and after the topology updates in Fig. 7. After RigL updates we observe new negative eigenvalues with significantly larger magnitudes. We also see larger positive eigenvalues, which disappear after few iterations[‖]. In comparison, the effect of Sparse Evolutionary Training (SET) updates on the Hessian spectrum seems limited.

We also evaluate the Hessian spectrum of LeNet5 during the training. In Fig. 8b, we observe similar shapes for each method on the positive side of the spectrum, however, on the negative side dense models seem to have more mass. We plot the magnitude of the largest negative eigenvalue

[‖] We share videos of these transitions in supplementary material.

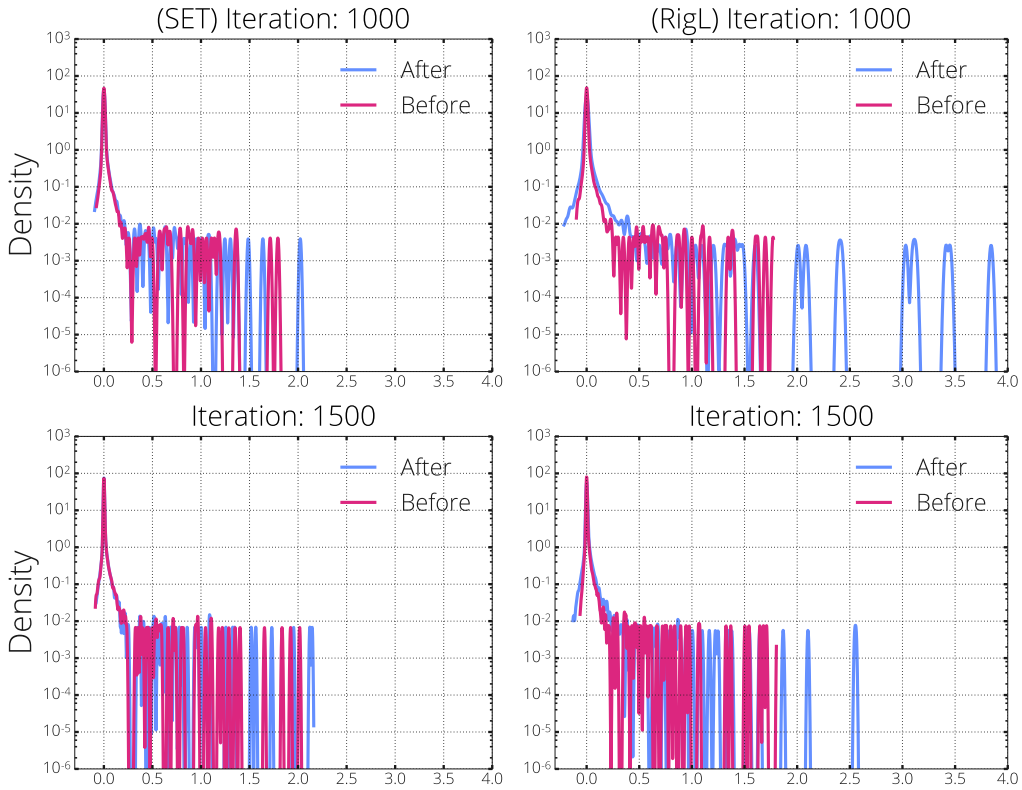


Figure 7: Hessian spectrum before and after mask updates: **(left)** SET **(right)** RigL. Similar to Ghorbani et al., 2019, we estimate the spectral density of Hessian using Gaussian kernels.

to characterize this behaviour in Fig. 8a. We observe a significant difference between sparse and dense models and observe that sparse networks trained with RigL have larger negative eigenvalues.

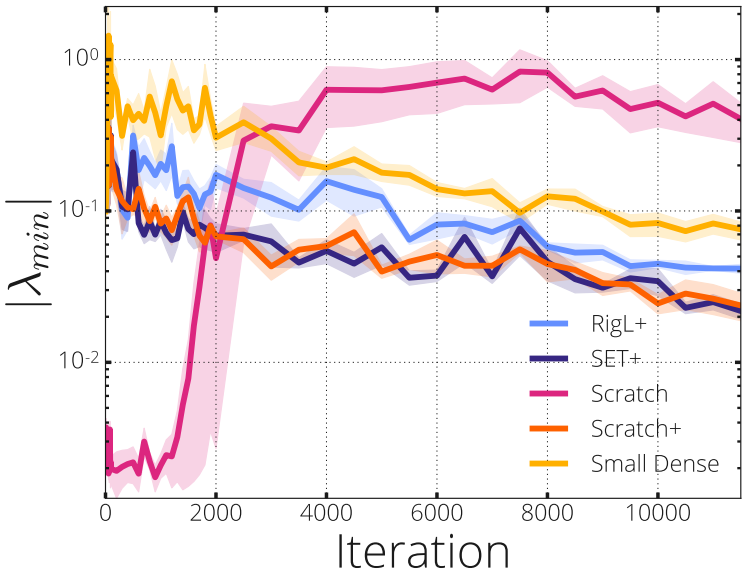
D Comparing Function Similarity

Table 5 gives a full list of comparison metrics of the predictions on the test set for LeNet5 on MNIST and ResNet50 on ImageNet-2012, in particular here we also compare the output probability distributions using relevant metrics.

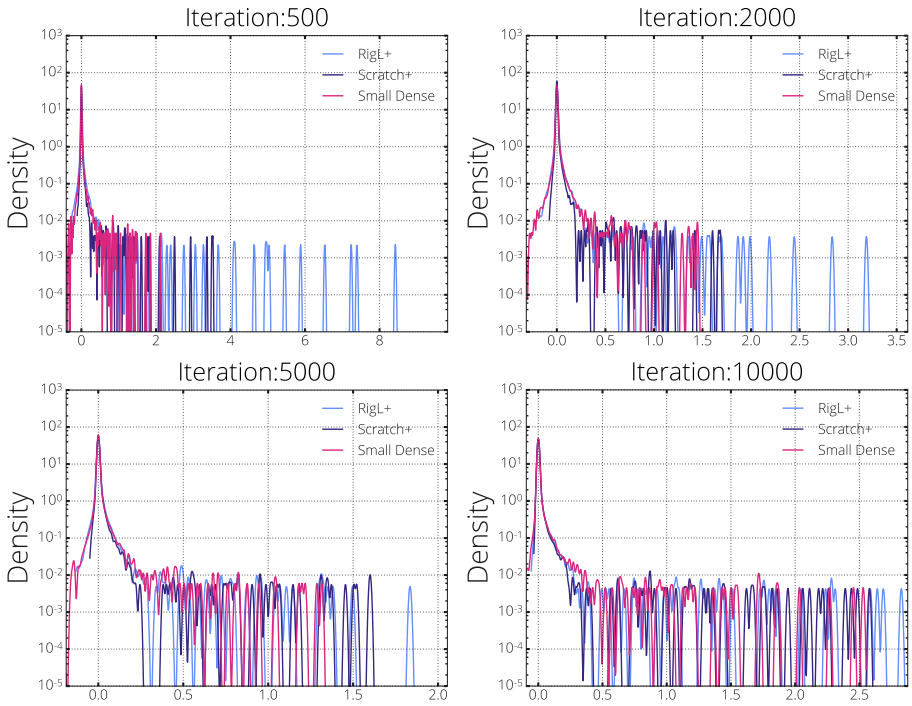
Table 5: **Ensemble/Prediction Disagreement**. In order to show the function similarity of LTs to the pruning solution, we follow the analysis of (Fort et al., 2020), and compare the function similarity and ensemble generalization over 5 sparse models trained using random initializations and LTs with the original pruning solution they are derived from. The fractional disagreement is the pairwise disagreement of class predictions over the test set, as compared within the group of sparse models, and as compared to the pruned model whose mask they were derived from. Kullback–Leibler Divergence (KL) and Jensen–Shannon Divergence (JSD) compare the prediction distributions over all the test samples.

MNIST/LeNet5								
Init. Method	**Test Acc. (Top-1)**	**Ensemble**	**Pairwise Disagree.**	**Disagree. w/ Pruned**	**Pairwise KL**	**KL w/ Pruned**	**5-model JSD**	**JSD w/ Pruned**
Pruned Soln.*	0.9828	–	–	–	–	–	–	–
LT	98.36 ±0.05	98.39	0.0057±0.0005	0.0096±0.0004	0.0126±0.0010	0.0232±0.0010	0.00358±0.03000	0.0045±0.0002
Scratch	90.36 ±9.79	96.30	0.1436±0.1128	0.0930±0.0974	1.59 ±1.50	1.14 ±1.50	0.173 ±0.300	0.0599±0.0700
Pruned w/ Diff. Init.**	98.100±0.210	98.70	0.0222±0.0020	0.0210±0.0020***	0.126 ±0.020		0.0220 ±0.1000	0.0124±0.0010
Init. Method	**Test Acc. (Top-1)**	**Ensemble**	**Pairwise Disagree.**	**Disagree. w/ Pruned**	**Pairwise KL**	**KL w/ Pruned**	**5-model JSD**	**JSD w/ Pruned**
ImageNet-2012/ResNet-50								
Pruned Soln.*	0.7554	–	–	–	–	–	–	–
LT	75.730±0.008	76.27	0.0894±0.0008	0.0941±0.0009	3325.0±26.4	3605.0±12.4	787.20±5.24	853.70±3.55
Scratch	71.16 ±0.13	74.05	0.2039±0.0013	0.2033±0.0012	15787 ±105	20442 ±178	3316.0 ±13.2	3847.0 ±26.0
Pruned w/ Diff. Init.**	75.65 ±0.13	77.78	0.1620±0.0008	0.1623±0.0011***	13158 ±143	13087.0±55.9	2760.0 ±16.8	2755.00±4.96

* This is the pruning solution that the LT and scratch models are derived from.
** 5 pruning solutions found with different random initialization, one of which is the pruning solution above.
*** Here we compare 4 different pruned models with the pruning solution the LT/Scratch are derived from.



(a)



(b)

Figure 8: MNIST Hessian spectrum experiments.