Are All Layers Created Equal?

Chiyuan Zhang
chiyuan@google.com

Samy Bengio
bengio@google.com

Yoram Singer
singer@google.com

May 28, 2019

Abstract

Understanding deep neural networks has been a major research objective in recent years with notable theoretical progress. A focal point of those studies stems from the success of excessively large networks which defy the classical wisdom of uniform convergence and learnability. We study empirically the layer-wise functional structure of overparameterized deep models. We provide evidence for the heterogeneous characteristic of layers. To do so, we introduce the notion of robustness to post-training *re-initialization* and *re-randomization*. We show that the layers can be categorized as either "ambient" or "critical". Resetting the ambient layers to their initial values has no negative consequence, and in many cases they barely change throughout training. On the contrary, resetting the critical layers completely destroys the predictor and the performance drops to chanceh. Our study provides further evidence that mere parameter counting or norm accounting is too coarse in studying generalization of deep models, and flatness or robustness analysis of the models needs to respect the network architectures.

1 Introduction

Deep neural networks have been remarkably successful in many real world machine learning applications. In critical applications, distilled understanding of the systems can be as important as achieving the state-of-the-art performance. One important question is on interpreting and explaining the decision function of trained networks. It is closely related to another important topic on networks' generalization and robustness under drifting or even adversarially perturbed data distribution. In this paper, we study how individual layers coordinate the computation in trained neural network models, and relate the empirical results to generalization and robustness properties.

Theoretical research of the functions computed by neural networks dates back to the '80s. It is known that a neural network with a single (sufficiently wide) hidden layer is a universal approximator for continuous functions over compact domains (Gybenko, 1989; Hornik, 1991; Anthony and Bartlett, 2009). More recent research further examines whether *deep* networks can have superior representation power than *shallow* ones with the same number of units or edges (Pinkus, 1999; Delalleau and Bengio, 2011; Montufar et al., 2014; Telgarsky, 2016; Shaham et al., 2015; Eldan and Shamir, 2015; Mhaskar and Poggio, 2016; Rolnick and Tegmark, 2017). The capacity to represent arbitrary functions on finite samples is also extensively discussed (Hardt and Ma, 2017; Zhang et al., 2017; Nguyen and Hein, 2018; Yun et al., 2018). However, the constructions used in the aforementioned work for building networks approximating particular functions are typically "artificial" and are unlikely to be obtained by gradient-based learning algorithms. We focus instead on empirically studying the role different layers take in representing a learned function *post* gradient-based training.

Generalization is a fundamental theoretical question in machine learning. The recent observation that big neural networks can fit random labels on the training set (Zhang et al., 2017) makes it difficult to apply classical learning theoretic results based on uniform convergence over the hypothesis space. One approach to get around this issue is to show that, while the space of neural networks of a given architecture is huge, gradient-based learning on "well behaved" tasks leads to relatively "simple" models. More recent research focuses on the analysis of the post-training complexity metrics such as *norm*, *margin*, *robustness*, *flatness*, or *compressibility* of the learned model in contrast to the pre-training *capacity* of the entire hypothesis space. This line of work resulted in improved generalization bounds for deep neural networks (e.g. Dziugaite and Roy, 2016; Kawaguchi et al., 2017; Bartlett et al., 2017; Neyshabur et al.,

2018, 2017; Liang et al., 2017; Arora et al., 2018; Zhou et al., 2019). This work provides further empirical evidence and alludes to more fine-grained analysis. We show that the layers in a deep network are not homogeneous in the role they play at representing a predictor. Some layers are critical to forming good predictions while others are ambient as they are fairly insensitive to the assignment of their weights during training. Thus, depending on the capacity of the network and the complexity of the target function, gradient-based trained networks conserve the complexity by not using excess capacity.

Before proceeding, we would like to further mention a few related papers. Modern neural networks are typically overparameterized and thus redundant in their representations. Previous work exploited overparameterization to compress (Han et al., 2015) or distill (Hinton et al., 2015) a trained network. It is also shown that one can achieve comparable performance by training only a small fraction of network parameters such as a subset of the channels in each convolutional layer Rosenfeld and Tsotsos (2018). As a tool for interpreting residual networks as ensemble of shallow networks, Veit et al. (2016) found that residual blocks in a trained network can be deleted or permuted to some extent without degrading the performance too much. Another line of research showed that under extreme overparameterization, such as when the network width is polynomial in the training set size and input dimension (Allen-Zhu et al., 2018; Du et al., 2018a,b; Zou et al., 2018), or even in the asymptotic regime of infinite width (Jacot et al., 2018; Lee et al., 2019), the network weights move slowly during training. We make similar observations in this paper. However, we find that in more pragmatic settings, different layers exhibit different behaviors and the network cannot be treated in a monolithic way.

The rest of the paper is organized as follows. Our experimental framework and notions of robustness to modifications of layers are introduced in Sec. 2. Sec. 3 presents the results and analysis of whole-layer robustness for a wide range of neural network models. Sec. 4 discusses the theoretical implications on generalization. Experiments with joint robustness and connections to other notions of robustness are presented in Sec. 5 and Sec. 6, respectively. Finally, the paper ends with a conclusion that summarize our main contributions.

2   Setting

Feed forward networks naturally consist of multiple *layers* where each unit in a layer takes inputs from units in the previous layer. Let $\mathcal{F}^D = \{f_\theta : \theta = (\theta_1, \dots, \theta_D)\}$ be the space of a particular neural network architecture with D (parametric) layers. We are interested in analyzing the *post-training* behavior of whole layers of common deep networks. Such networks are typically trained using stochastic gradient methods which initialize the parameters by sampling from a pre-defined distribution $\theta_d^0 \sim \mathcal{P}_d$. The choice of $\mathcal{P}_d$ depends on the type, fan-in, and fan-out of each layer. After training for T epochs, the parameters of the last epoch θ^T are used as the final trained model. We save the model parameters at each epoch during training as *checkpoints*. Checkpoint-0 contains random weights initialized *before* seeing the training data, and checkpoint-T contains the weights for the final model.

A deep network builds up the *representation* of its inputs by incrementally applying nonlinear transformations defined by each layer. As a result, the representation at a particular layer recursively depends on all the layers beneath it. This complex dependency makes it challenging to isolate and inspect each layer independently in theoretical studies. In this paper, we introduce and use the following two empirical probes to inspect the individual layers in a trained neural network.

Re-initialization   After training concludes, for a given layer $d = 1, \dots, D$, we *re-initialize* the parameters through assignment $\theta_d^T \leftarrow \theta_d^0$, while keeping the parameters for the other layers intact. The model with the parameters $(\theta_1^T, \dots, \theta_{d-1}^T, \theta_d^0, \theta_{d+1}^T, \dots, \theta_D^T)$ is then evaluated. Unless noted otherwise, we use the term *performance* to designate the test performance measured by the 0-1 classification loss. The performance of a network in which layer d was re-initialized is referred to as the *re-initialization robustness* of layer d. Note that here θ_d^0 denotes the random values loaded from checkpoint-0. More generally, for k epochs $\tau_1, \dots, \tau_k \in [0, T]$, we can *re-initialize* the d-th layer by setting $\theta_d^T \leftarrow \theta_d^\tau$, and obtain the *re-initialization robustness* of layer d for checkpoint-τ.

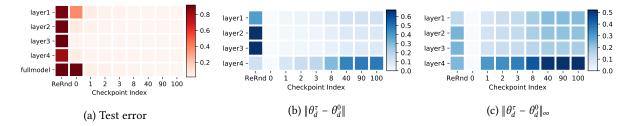| | (a) Test error | (b) $\|\theta_d^\tau - \theta_d^0\|$ | (c) $\|\theta_d^\tau - \theta_d^0\|_\infty$ |

Figure 1: **Robustness results for** FCN3 × 256 **on MNIST.** (a) Test error rate: each row corresponds to one layer in the network. The last row shows full model's performance (i.e. model parameters are loaded from that checkpoint) for the corresponding epoch as reference. The first column designates robustness of each layer w.r.t re-randomization and the rest of the columns designate re-initialization robustness at different checkpoints. The last column shows the final performance (at the last checkpoint during training) for reference. (b-c) Weights distances: each cell in the heatmaps depict the normalized 2-norm (b) or ∞-norm (c) distance of trained parameters to their initial weights.

Re-randomization To go one step further, *re-randomization* of a layer d means re-sampling random values $\tilde{\theta}_d \sim \mathcal{P}_d$ and evaluating the model's performance for $(\theta_1^T, \ldots, \theta_{d-1}^T, \tilde{\theta}_d, \theta_{d+1}^T, \ldots, \theta_D^T)$. Analogously, we refer to the evaluated performance as the *re-randomization robustness* of layer d.

Note that *no* re-training or fine-tuning after re-initialization or re-randomization is conducted and the network is evaluated directly with mixed post-trained and re-initialized/re-randomized weights. When a network exhibits negligible decrease[1] in performance after re-initializing or re-randomizing of a layer, we say that the layer is *ambient*, and otherwise the layer is called *critical*.

3 Robustness of individual layers

The datasets we use in the robustness studies are standard image classification benchmarks: MNIST, CIFAR10, and ImageNet. All the networks are trained using SGD with momentum, and piecewise constant learning rate schedule, see Appendix A for further details.

3.1 Fully connected networks

We start by examining the robustness of fully-connected networks (FCN). A FCN $D \times H$ consists of D fully connected layers each of output dimension H followed by the ReLU activation function. The additional final layer is a linear multiclass predictor with one output per class.

As a starter, we train an FCN 3 × 256 on MNIST, and apply the re-initialization and re-randomization analysis on the trained model. The results are shown in Fig. 1(a). As expected, due to the intricate dependency of the classification function on each of the layers, re-randomizing any of the layers completely disintegrates the representation and classification accuracy drops to the level of random guessing. However, for re-initialization, while the first layer is very sensitive, the rest of the layers are robust to re-initialization to their (pre-training) random weights.

A plausible explanation for this could be attributed to that the gradient norms increase during back-propagation such that the bottom layers are being updated more aggressively than the top ones. However, if this were the case, we would expect a smoother transition instead of a sharp contrast at the first layer. Furthermore, we measured how distant the weights of each layer are from their initialization, "checkpoint-0", using both the 2-norm (normalized by $1/\sqrt{\text{num params}}$) and the ∞-norm. The results are shown in Fig. 1(b) and (c), respectively. As we can see, the robustness to re-initialization does not obviously correlate with either of the distances. This suggests that there might

---

[1]There is no universal threshold to quantify how much is "negligible" across all models and tasks. But as we will see from the empirical results, there is no ambiguity in identifying the layer types, due to the sharp performance contrast between the ambient and critical layers.
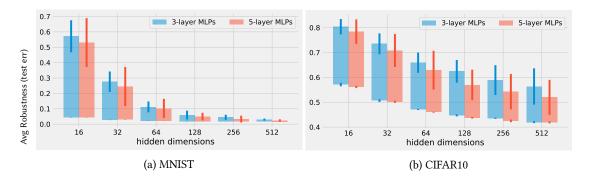
| (a) MNIST | (b) CIFAR10 |

Figure 2: **Average re-initialization robustness to checkpoint-0 of all layers but the first for** FCNs. Each bar designates the difference in classification error between a model with one layer re-initialized (top of bar) and the same model without weight modification (bottom of bar). The error-bars designate one standard deviation obtained by running five experiments with different random initializations.

be something more intricate going on than simple gradient expansion. We informally summarize the observations as follows,

> *Over-capacitated deep networks trained with stochastic gradient have low-complexity due to self-restriction of the number of critical layers.*

Intuitively, if a subset of parameters can be re-initialized to the random values at checkpoint-0 (which are independent of the training data), then the effective number of parameters, and as a result, the complexity of the model, can be reduced.

We apply the same analysis framework to a large number of different configurations to assess the influence of the network capacity and the task complexity on the layer robustness. In Fig. 2(a), we compare the average re-initialization robustness for all layers but the first with respect FCNs of varying hidden dimensions on MNIST. It is clear that the top layers become more robust as the hidden dimension increases. We believe that it reflects the fact that the wider FCNs have higher model capacity. When the capacity is small, all layers are vigil participants in representing the prediction function. As the capacity increases, it suffices to use the bottom layer while the rest act as random projections with non-linearities.

Similarly, Fig. 2(b) shows experiments on CIFAR10, which has the same number of classes and comparable number of training examples as MNIST, but is more difficult to classify. We observe similar traits as the hidden dimensions increase, though not as pronounced as in MNIST. Informally put, the difficulty of the learning task seem to necessitate more diligence of the layers in forming accurate predictors.

In summary, the empirical results of this section provide some evidence that deep networks *automatically* adjust their de-facto capacity. When a big network is trained on an easy task, only a few layers seem to be playing critical roles.

3.2 Large convolutional networks

In typical computer vision tasks beyond MNIST, densely connected FCNs are outperformed significantly by convolutional neural networks. VGGs and ResNets are among the most widely benchmarked convolutional network architectures. Fig. 3 and Fig. 4 show the robustness results on CIFAR10 for the two architectures, respectively.

Since the networks are much deeper than FCNs, we transpose the heatmaps to show the layers as columns. For VGGs, more layers are sensitive to re-initialization, yet the patterns are similar to the observations from the simple FCNs on MNIST: the bottom layers are sensitive while the top layers are robust to re-initialization.

The results for ResNets in Fig. 4 are to be considered together with results on ImageNet in Fig. 5. We found the robustness patterns for ResNets more interesting for the following reasons.

(a) VGG11
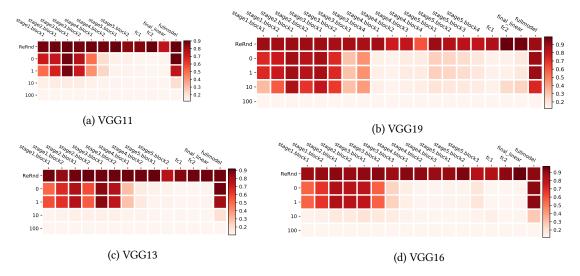
(b) VGG19

(c) VGG13

(d) VGG16

Figure 3: **Whole-layer robustness for** VGG **networks on CIFAR10.** Heatmaps use the same layout as in Fig. 1 after being transposed to visualize the deeper architecture more effectively.

ResNets re-distribute critical layers. Unlike the FCNs and VGGs which place the critical layers at the bottom of the network, ResNets distribute them across the network. To better understand the patterns, let us briefly recap the ResNet architecture commonly used in practice. Morally, a ResNet is divided into "stages". At the bottom, there is a pre-processing stage (`stage0`) with vanilla convolutional layers. It is followed by a few (typically 4) residual stages consisting of multiple residual blocks, and finally a global average pooling and a densely connected linear classifier (`final_linear`). The image size halves and the number of convolution channels doubles from each residual stage to the next one[2]. As a result, while most of the residual blocks have real *identity* skip connections, the first block of each stage (`stage*.resblk1`), which is connected to the last block of the previous stage, has a *non-identity* skip connection due to different input-output shapes. Fig. 7 in the Appendix illustrates the two types of residual blocks. In our robustness analysis, we can interpret each stage of a ResNet as a sub-network, with characteristics of whole-layer robustness *within* each stage similar to VGGs or FCNs.

Residual blocks can be robust to re-randomization. Among the layers that are robust to re-initialization, if the layer is a residual block, it is also robust to re-randomization, which stands in contrast to the `final_linear` layer. This could be potentially attributed to the fact that the identity skip connection dominates the residual branches in those blocks. It is known from previous research (Veit et al., 2016) that residual blocks in a ResNet can be removed without substantially hurting accuracy. Our experiments have a different focus as they study robustness in the light of the interplay between model capacity and difficulty of the learning task. In particular, comparing the results on the two different datasets, especially on smaller ResNets (e.g. ResNet18), many residual blocks with real identity skip connection also become sensitive on the more challenging ImageNet task.

4 Implications on generalization

As mentioned above, if some parameters can be re-assigned their initial values without hurting model's performance, then the effective number of parameters is reduced as the random weights are dissociated from the training data. The benefits on improving generalization can be naively demonstrated using parameter counting in standard generalization bounds. For example, if we have a generalization bound of the form

$$R\left(\hat{f}_n^m\right) \le \hat{R}_n\left(\hat{f}_n^m\right) + \mathcal{B}(m, n),$$

---

[2]There are more subtle details especially at `stage1` depending on factors like the input size, whether residual blocks contain a bottleneck, etc.

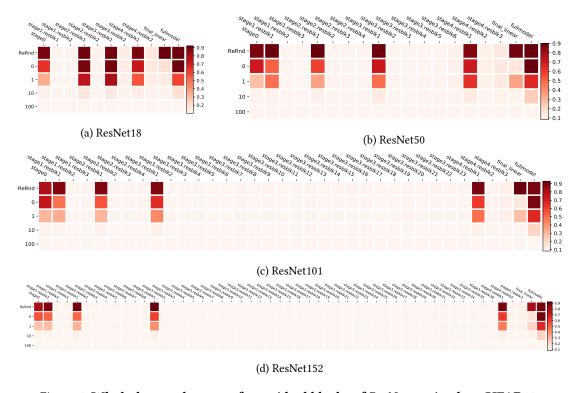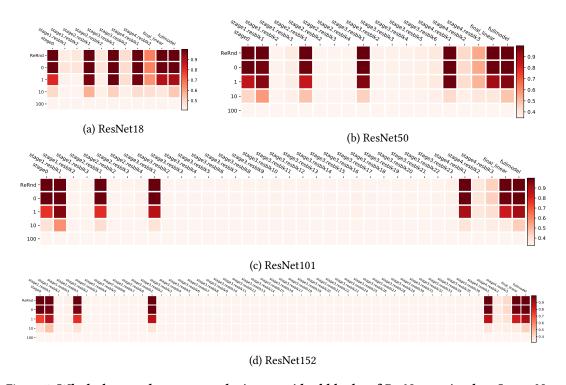Figure 4: **Whole-layer robustness for residual blocks of** ResNet**s trained on CIFAR10.**

Figure 5: **Whole-layer robustness analysis on residual blocks of** ResNet**s trained on ImageNet.**

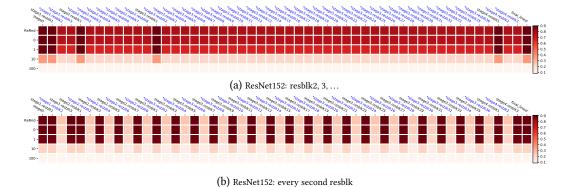(a) ResNet152: resblk2, 3, ...

(b) ResNet152: every second resblk

Figure 6: **Joint robustness of** ResNet**152 on CIFAR10**. Jointly re-initialized/re-randomized layer groupings are indicated with the ∗ over the layer names (also colored as blue for easier identification).

where $\hat{f}_n^m$ represents a model with m parameters trained on n i.i.d. samples, and $\mathcal{B}$ is a generalization bound based on parameter counting. For example, Anthony and Bartlett (2009) provided various bounds on VC-dimension based on the number of weights of a neural network. This bound can be further used in standard VC-based generalization bounds for classification (Vapnik, 1998). Now, if we know *a-priori* that a fraction $\rho \in (0, 1)$ of the network's weights will be robust to re-initialization after training with degradation of empirical risk by at most ε, we then get

$$R\left(\hat{f}_n^{(1-\rho)m}\right) \le \hat{R}_n\left(\hat{f}_n^m\right) + \varepsilon + \mathcal{B}((1-\rho)m, n).$$

Here $\hat{f}_n^{(1-\rho)m}$ is a model obtained by re-initializing ρ fraction of parameters of the trained model $\hat{f}_n^m$. Note that generalization bounds based on parameter counting are typically not meaningful in the context of deep learning. Due to heavy overparameterization, the resulting bounds are usually vacuous. However, as noted in Arora et al. (2018), most of the alternative generalization bounds proposed recently for deep networks are actually worse than naive parameter counting. Moreover, by tweaking existing analyses with additional whole-layer robustness condition, some PAC-Bayes based bounds can also be potentially improved (Wang et al., 2018; Arora et al., 2018; Zhou et al., 2019).

The bounds are applicable when we know which layers are ambient *prior* to obtaining the training data. We observed strong patterns of the distribution of ambient layers, thus we can usually identify ambient layers purely from the network architectures. Furthermore, we can also propose K different guesses of ambient layers, and the best generalization bound out of the K cases can be obtained via elementary union bound, incurring a factor of $\mathcal{O}(\log(K))$ on the bound. As the results in Arora et al. (2018); Zhou et al. (2019), the bounds provided by re-initialization robustness are for a different model, in our case the re-initialized one. Alternative approaches in the literature involve modifying the training algorithms to explicitly optimize the robustness or some derived generalization bounds (Neyshabur et al., 2015; Dziugaite and Roy, 2016).

5 Joint robustness

The empirical results we presented thus far focus on *whole-layer* robustness. We next explore *joint robustness* of multiple layers through *simultaneous* re-initialization or re-randomization .

We divide the layers into two groups and perform robustness experiments with each group. Fig. 6(a) demonstrate that naively grouping all the ambient layers into one group does not yield good performance. However, an alternative grouping scheme shown in Fig. 6(b) demonstrates that robustness can be significantly improved when jointly resetting about half of the layers for this ResNet. See Appendix B for more details. Note SGD fits the model with the aforementioned joint robustness structure without explicit constraints. We next examine how explicit constraints could improve robustness. Concretely, we experiment with two approaches: i. Refrain from training layers and leave their parameters at the randomly initialized values. ii. Remove layers from the network.

Table 1: **Error rates (%) on CIFAR10 (top) and ImageNet (bottom).** Each row reports the performance of a full model, whole-layer robustness to re-initialization (mean±std), partially trained models with a subset of the layers stranded to their initial values, partially train model after removing a subset of the layers. Whole-layer robustness is averaged over all the residual blocks except for the first at each stage. Layer-freezing and layer-removal are jointly applied to those residual blocks.

	Arch	Full Model	Whole-layer Robustness	Layers Frozen	Layers Removed
CIFAR10	ResNet50	8.40	9.77±1.38	11.74	9.23
	ResNet101	8.53	8.87±0.50	9.21	9.23
	ResNet152	8.54	8.74±0.39	9.17	9.23
ImageNet	ResNet50	34.74	38.54±5.36	44.36	41.50
	ResNet101	32.78	33.84±2.10	36.03	41.50
	ResNet152	31.74	32.42±1.55	35.75	41.50

The results are given in Table 1. When we freeze some layers, the test error is higher than the average whole-layer robustness measured in a normally trained model. However, the gap is much smaller than directly measuring the joint robustness. Moreover, on CIFAR10, we find that similar performance can be achieved even if we remove entirely those layers from the network. In contrast, for ImageNet layer removal results in a significant performance gap. In this case, random projections followed by non-linear activations conducted by frozen layers deem necessary to maintain the accuracy.

6   Connections to other notions of robustness

The notion of whole-layer and joint robustness to re-initialization and re-randomization can be related to other notions of robustness in deep learning. For example, *flatness* refers to robustness to *local* perturbations of the network's parameters at convergence, and is extensively discussed in the context of generalization (Hochreiter and Schmidhuber, 1997; Chaudhari et al., 2017; Keskar et al., 2017; Smith and Le, 2018; Poggio et al., 2018). For a fixed layer, our notion of robustness to re-initialization is restricted to be on the optimization trajectory, which could potentially take the form of *non-local* perturbations. Robustness to re-randomization allows for further large variances of perturbations to the trained parameters. As our study shows, robustness seems to be layer dependent, thus analyzing layers individually for specific network architectures allows us to obtain further insights to the robustness behaviors.

In contrast, *adversarial* robustness (Szegedy et al., 2013) focuses on robustness to perturbations of the input. In particular, it was found that trained deep networks are sensitive to small adversarial perturbations which yield prediction shifts to to arbitrary classes. A large number of defense and attack algorithms have been proposed in recent years along this line. Here we briefly discuss the connection to adversarial robustness. Take a normally trained ResNet with S stages with $(B_1, \dots, B_S)$ residual blocks in each stage. At test time, we turn it into a stochastic classifier by randomly selecting a subset of $s \in [0, S]$ stages, and randomly replacing a residual block from each of the selected stage with one of the r pre-initialized weights of its layer. We keep r pre-allocated weights for each residual block instead of re-sampling at random on each evaluation call, primarily to reduce the computation burden during the test phase.

From the robustness analysis in the previous sections, we expect the stochastic classifier to get only a small *average* performance drop. However, at individual example level, the randomness of the network outputs will make it harder for the attacker to generate adversarial examples. We evaluate the adversarial robustness against a weak FGSM (Goodfellow et al., 2014) attack and a strong PGD (Madry et al., 2017) attack. The results in Table 2 show that, compared to the baseline (the exact same trained model without stochastic evaluation), the randomness significantly increases the adversarial robustness against weak attacks. The performances under strong PGD attack drop to very low, but still with a non-trivial gap between the baseline.

In summary, whole-layer robustness could improve the adversarial robustness of a trained model through injected

Table 2: **Accuracy (%) of various model configurations on clean CIFAR10 test set, under weak (FGSM), and strong (PGD) adversarial attack.** Adversarial attacks are evaluated on a subset of 1000 test examples. Every experiment is repeated 5 times and the average performance is reported. The hyperparameters r and s in model configurations correspond to the number of random weights pre-set for each residual block, and the number of stages that are re-randomized during each inference. Here 4^2 designates a ResNet architecture with two stages, each stage of four residual blocks, Similarly, 4^4 network has four stages each with four residual blocks.

Model Configuration		Clean	FGSM	PGD
4^2	baseline	91.05 ± 0.00	12.75 ± 0.04	0.33 ± 0.16
	r=4,s=1	89.45 ± 0.13	69.85 ± 1.60	6.71 ± 0.37
	r=4,s=2	87.70 ± 0.25	71.18 ± 0.49	9.65 ± 0.27
4^4	baseline	90.08 ± 0.00	8.45 ± 0.00	0.00 ± 0.00
	r=4,s=1	89.64 ± 0.12	62.76 ± 1.09	2.60 ± 0.26
	r=4,s=2	89.13 ± 0.13	67.20 ± 0.63	3.56 ± 0.48
	r=4,s=4	88.24 ± 0.18	69.09 ± 1.59	5.60 ± 0.53

stochasticity. However, it is does not constitute sufficient defence against strong attackers. More sophisticated attacks that explicitly deal with stochastic classifiers might completely break this model.

7 Conclusions

We investigated the functional structure on a layer-by-layer basis of overparameterized deep models, on a wide variety of popular models for image classification. We introduced the notions of re-initialization and re-randomization robustness. Using these notions we provided evidence for the heterogeneous characteristic of layers, which can be categorized into either ambient or critical. Resetting the ambient layers to their initial value has negligible consequence on the model's performance. Our empirical results give further evidence that mere parameter counting or norm accounting is too coarse in studying generalization of deep models. Moreover, optimization landscape based analysis is better performed respecting the network architectures due to the heterogeneous behaviors of different layers. For future work, we are interested in devising a new algorithm which learns the interleaving trained and partially random subnetworks within one large network.

References

Allen-Zhu, Z., Li, Y., and Song, Z. (2018). A convergence theory for deep learning via Over-Parameterization. *CoRR*, arXiv:1811.03962.

Anthony, M. and Bartlett, P. L. (2009). *Neural Network Learning: Theoretical Foundations*. Cambridge University Press.

Arora, S., Ge, R., Neyshabur, B., and Zhang, Y. (2018). Stronger generalization bounds for deep nets via a compression approach. *CoRR*, arXiv:1802.05296.

Bartlett, P. L., Foster, D. J., and Telgarsky, M. J. (2017). Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, pages 6240–6249.

Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., Chayes, J., Sagun, L., and Zecchina, R. (2017). Entropy-sgd: Biasing gradient descent into wide valleys. In *ICLR*.

Delalleau, O. and Bengio, Y. (2011). Shallow vs. Deep Sum-Product Networks. In *NIPS*, pages 666–674.

Du, S. S., Lee, J. D., Li, H., Wang, L., and Zhai, X. (2018a). Gradient descent finds global minima of deep neural networks. *CoRR*, arXiv:1811.03804.

Du, S. S., Zhai, X., Poczos, B., and Singh, A. (2018b). Gradient descent provably optimizes over-parameterized neural networks. *CoRR*, arXiv:1810.02054.

Dziugaite, G. K. and Roy, D. M. (2016). Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In *UAI*.

Eldan, R. and Shamir, O. (2015). The Power of Depth for Feedforward Neural Networks. *CoRR*, arXiv:1512.03965.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *CoRR*, arXiv:1412.6572.

Gybenko, G. (1989). Approximation by superposition of sigmoidal functions. *Mathematics of Control, Signals and Systems*, 2(4):303–314.

Han, S., Mao, H., and Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *CoRR*, arXiv:1510.00149.

Hardt, M. and Ma, T. (2017). Identity matters in deep learning. In *ICLR*.

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

He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *CoRR*, arXiv:1503.02531.

Hochreiter, S. and Schmidhuber, J. (1997). Flat minima. *Neural Computation*, 9(1):1–42.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

Jacot, A., Gabriel, F., and Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8580–8589.

Kawaguchi, K., Kaelbling, L. P., and Bengio, Y. (2017). Generalization in deep learning. *CoRR*, arXiv:1710.05468.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2017). On large-batch training for deep learning: Generalization gap and sharp minima. In *ICLR*.

Lee, J., Xiao, L., Schoenholz, S. S., Bahri, Y., Sohl-Dickstein, J., and Pennington, J. (2019). Wide neural networks of any depth evolve as linear models under gradient descent. *arXiv preprint arXiv:1902.06720*.

Liang, T., Poggio, T., Rakhlin, A., and Stokes, J. (2017). Fisher-rao metric, geometry, and complexity of neural networks. *CoRR*, arXiv:1711.01530.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks. *CoRR*, arXiv:1706.06083.

Mhaskar, H. and Poggio, T. A. (2016). Deep vs. shallow networks : An approximation theory perspective. *CoRR*, arXiv:1608.03287.

Montufar, G. F., Pascanu, R., Cho, K., and Bengio, Y. (2014). On the number of linear regions of deep neural networks. In *Advances in neural information processing systems (NIPS)*, pages 2924–2932.

Neyshabur, B., Bhojanapalli, S., McAllester, D., and Srebro, N. (2017). Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 5947–5956.

Neyshabur, B., Bhojanapalli, S., and Srebro, N. (2018). A PAC-Bayesian approach to Spectrally-Normalized margin bounds for neural networks. In *ICLR*.

Neyshabur, B., Salakhutdinov, R., and Srebro, N. (2015). Path-sgd: Path-normalized optimization in deep neural networks. In *NIPS*, pages 2422–2430.

Nguyen, Q. and Hein, M. (2018). Optimization Landscape and Expressivity of Deep CNNs. In *International Conference on Machine Learning*, pages 3727–3736.

Pinkus, A. (1999). Approximation theory of the MLP model in neural networks. *Acta Numerica*, 8:143–195.

Poggio, T., Liao, Q., Miranda, B., Banburski, A., Boix, X., and Hidary, J. (2018). Theory iiib: Generalization in deep networks. Technical report, MIT.

Rolnick, D. and Tegmark, M. (2017). The power of deeper networks for expressing natural functions. *CoRR*, arXiv:1705.05502.

Rosenfeld, A. and Tsotsos, J. K. (2018). Intriguing Properties of Randomly Weighted Networks: Generalizing While Learning Next to Nothing. *CoRR*, arXiv:1802.00844.

Shaham, U., Cloninger, A., and Coifman, R. R. (2015). Provable approximation properties for deep neural networks. *CoRR*, arXiv:1509.07385.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Smith, S. L. and Le, Q. V. (2018). A bayesian perspective on generalization and stochastic gradient descent. In *ICLR*.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *CoRR*, arXiv:1312.6199.

Telgarsky, M. (2016). benefits of depth in neural networks. In Feldman, V., Rakhlin, A., and Shamir, O., editors, *29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pages 1517–1539, Columbia University, New York, New York, USA. PMLR.

Vapnik, V. N. (1998). *Statistical Learning Theory*. Adaptive and learning systems for signal processing, communications, and control. Wiley.

Veit, A., Wilber, M. J., and Belongie, S. (2016). Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems*, pages 550–558.

Wang, H., Keskar, N. S., Xiong, C., and Socher, R. (2018). Identifying Generalization Properties in Neural Networks. *CoRR*, arXiv:1809.07402.

Yun, C., Sra, S., and Jadbabaie, A. (2018). Finite sample expressive power of small-width relu networks. *CoRR*, arXiv:1810.07770.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2017). Understanding deep learning requires rethinking generalization. In *ICLR*.

Zhou, W., Veitch, V., Austern, M., Adams, R. P., and Orbanz, P. (2019). Non-vacuous generalization bounds at the ImageNet scale: a PAC-Bayesian compression approach. In *ICLR*.

Zou, D., Cao, Y., Zhou, D., and Gu, Q. (2018). Stochastic gradient descent optimizes over-parameterized deep ReLU networks. *CoRR*, arXiv:1811.08888.

Appendix to "Are All Layers Created Equal?"

A   Details on experiment setup

Our empirical studies are based on the MNIST, CIFAR10 and the ILSVRC 2012 ImageNet datasets. Stochastic Gradient Descent (SGD) with a momentum of 0.9 is used to minimize the multi-class cross entropy loss. Each model is trained for 100 epochs, using a stage-wise constant learning rate scheduling with a multiplicative factor of 0.2 on epoch 30, 60 and 90. Batch size of 128 is used, except for ResNets with more than 50 layers on ImageNet, where batch size of 64 is used due to device memory constraints.

We mainly study three types of neural network architectures:

- FCNs: the FCNs consist of fully connected layers with equal output dimension and ReLU activation (except for the last layer, where the output dimension equals the number of classes and no ReLU is applied). For example, FCN 3 × 256 has three layers of fully connected layers with the output dimension 256, and an extra final (fully connected) classifier layer with the output dimension 10 (for CIFAR10 and MNIST).

- VGGs: widely used network architectures from Simonyan and Zisserman (2014), consist of multiple convolutional layers, followed by multiple fully connected layers and the final linear classifier layer.

- ResNets: the results from our analysis are similar for ResNets V1 (He et al., 2016a) and V2 (He et al., 2016b). We report our results with ResNets V2 due to the slightly better performance in most of the cases. For large image sizes from ImageNet, the `stage0` contains a 7 × 7 convolution and a 3 × 3 max pooling (both with stride 2) to reduce the spatial dimension (from 224 to 56). On smaller image sizes like CIFAR10, we use a 3 × 3 convolution with stride 1 here to avoid reducing the spatial dimension. Fig. 7 illustrates the two types of residual blocks that are used inside (with an *identity* skip connection) and between (with a *downsample* skip connection) stages.

  In the experiments on adversarial robustness in Sec. 6, we use a slightly modified variant by explicitly having a downsample layer between stages, so that all the residual blocks are with *identity* skip connections.

  The ResNets used in the main text are *without* batch normalization. Please see Appendix C for details and full comparison of the architectures without and without batch normalization.

During training, CIFAR10 images are padded with 4 pixels of zeros on all sides, then randomly flipped (horizontally) and cropped. ImageNet images are randomly cropped during training and center-cropped during testing. Global mean and standard deviation are computed on all the training pixels and applied to normalize the inputs on each dataset.
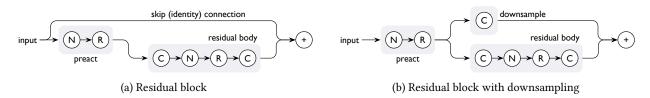

(a) Residual block   (b) Residual block with downsampling

Figure 7: **Illustration of residual blocks (from** ResNet**s V2) with and without a downsampling skip branch.** C, N and R stand for convolution, (batch) normalization and ReLU activation, respectively. Those are *basic* residual blocks used in ResNet18 and ResNet34; for ResNet50 and more layers, the *bottleneck* residual blocks are used, which are similar to the illustrations here except the residual body is now C → N → R → C → N → R → C with a 4× reduction of the convolution channels in the middle for a "bottlenecked" residual.

(a) layer2~6

(b) layer2,3,5,6

(c) layer2,4,6
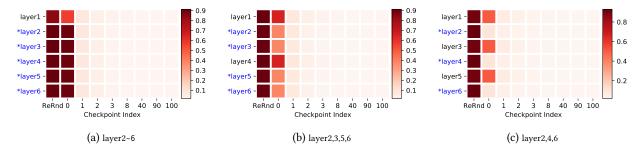
Figure 8: **Joint robustness analysis of** FCN 5 × 256 **on MNIST.** The heatmap layout is the same as in Fig. 1, but the layers are divided into two groups (indicated by the * mark on the blue colored layer names in each figure) and re-randomization and re-initialization are applied to all the layers in each group *jointly*. As a result, layers belonging to the same group have identical rows in the heatmap, but we still show all the layers to make the figures easier to compare with the previous *whole-layer* robustness results. The subfigures show the results from three different grouping schemes.

B   Further details on joint robustness

In this appendix, we provide results on joint robustness analysis that were not included in the main text due to space limit. From Sec. 3.1, we see that on MNIST, for wide enough FCNs, all the layers above `layer1` are robust to re-initialization. So we divide the layer into two groups: {`layer1`} and {`layer2`, `layer3`, …}, and perform the robustness studies on the two groups. The results for FCN 5 × 256 are shown in Fig. 8(a). For clarity and ease of comparison, the figure still spells out all the layers individually, but the values from `layer2` to `layer6` are simply repeated rows. The values show that the upper-layer-group is clearly *not* jointly robust to re-initialization (to checkpoint 0).

We also try some alternative grouping schemes: Fig. 8(b) show the results when we group two in every three layers, which has slightly improved joint robustness; In Fig. 8(c), the grouping scheme that include every other layer shows that with a clever grouping scheme, about half of the layers could be *jointly* robust.

Results on ResNets are similar. Fig. 9 shows the joint robustness analysis on ResNets trained on CIFAR10. The grouping is based on the whole-layer robustness results from Fig. 4: all the residual blocks in `stage1` to `stage4` are bundled and analyzed jointly. The results are similar to the FCNs: ResNet18 is relatively robust, but deeper ResNets are *not* jointly robust under this grouping. Two alternative grouping schemes are shown in Fig. 10. By including only layers from `stage1` and `stage4`, slightly improved robustness could be obtained on ResNet50. The scheme that groups every other residual block shows further improvements.

In summary, the individually robust layers are generally not jointly robust. But with some clever way of picking out a subset of the layers, joint robustness could still be achieved for up to half of the layers. In principle, one can enumerate all possible grouping schemes to find the best with a trade-off of the robustness and number of layers included.

C   Batch normalization and weight decay

The primary goal of this paper is to study the (co-)evolution of the representations at each layer during training and the robustness of this representation with respect to the rest of the network. We try to minimize the factors that explicitly encourage changing of the network weights or representations in the analysis. In particular, unless otherwise specified, weight decay and batch normalization are *not* used. This leads to some performance drop in the trained models. Especially for deep residual networks on ImageNet: even though we could successfully train a residual network with 100+ layers without batch normalization, the final generalization performance could be quite worse than the state-of-the-art. Therefore, in this section, we include studies on networks trained *with* weight decay

13

(a) ResNet18: resblk2

(b) ResNet50: resblk2, 3, …

(c) ResNet101: resblk2, 3, …
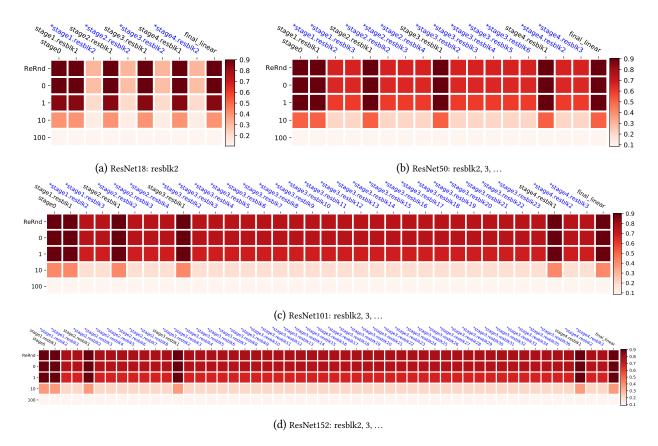
(d) ResNet152: resblk2, 3, …

Figure 9: **Joint robustness analysis of** ResNet**s on CIFAR10**, based on the scheme that group all but the first residual blocks in all the stages. Grouping is indicated by the ⋆ on the (blue colored) layer names.

and batch normalization for comparison.

Table 3 shows the final test error rates of models trained with or without weight decay and batch normalization. Note the original VGG models do not use batch normalization (Simonyan and Zisserman, 2014), we list +bn variants here for comparison, by applying batch normalization to the output of each convolutional layer. On CIFAR10, the performance gap varies from 3% to 5%, but on ImageNet, the gap could be as large as 10%.

Fig. 11 shows how different training configurations affect the whole-layer robustness analysis patterns on VGG16 networks. Fig. 12 and Fig. 13 show similar comparisons for ResNet50 on CIFAR10 and ImageNet, respectively. We found that the whole-layer robustness patterns are still quite pronounced under various training conditions. In Fig. 12(d) and Fig. 13(c,d), we found that re-initialing with checkpoint-1 is less robust than with checkpoint-0 for many layers. It might be that during early stages, some aggressive learning is causing changes in the parameters or statistics with large magnitudes, but later on when most of the training samples are classified correctly, the network gradually re-balances the layers to a more robust state. Fig. 15(d-f) in the next section show supportive evidence that, in this case the distance of the parameters between checkpoint-0 and checkpoint-1 is larger than between checkpoint-0 and the final checkpoint. However, on ImageNet this correlation is no longer clear as in Fig. 16(d-f). See also the discussions in the next section.

D   Robustness and distances

In Fig. 1 from Sec. 3.1, we compared the whole-layer robustness patterns to the layer-wise distances of the parameters to the values at initialization (checkpoint-0). We found that for FCNs on MNIST, there is no obvious correlation between the "amount of parameter updates received" at each layer and its robustness to re-initialization for the
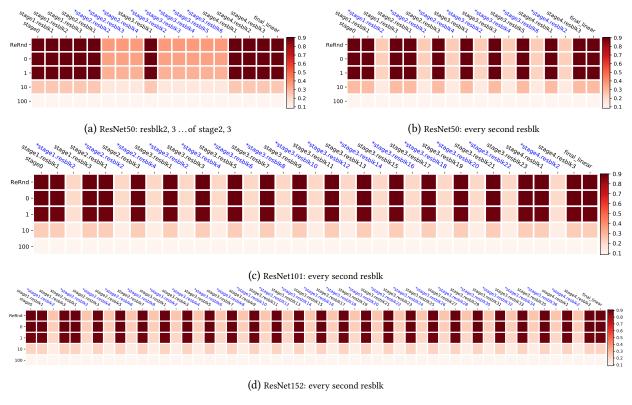
(a) ResNet50: resblk2, 3 ... of stage2, 3

(b) ResNet50: every second resblk

(c) ResNet101: every second resblk

(d) ResNet152: every second resblk

Figure 10: **Joint robustness analysis of** ResNet**s on CIFAR10**, with alternative grouping schemes. Grouping is indicated by the ⋆ on the (blue colored) layer names.

Table 3: **Test performance (classification error rates %) of various models studied in this paper.** The table shows how much of the final performance is affected by training with or without weight decay (+wd) and batch normalization (+bn).

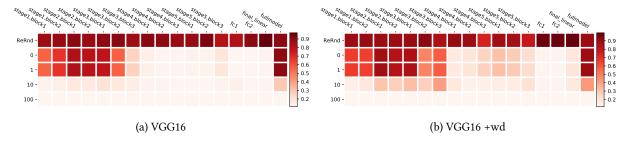|	Architecture	N/A	+wd	+bn	+wd+bn
CIFAR10	ResNet18	10.4	7.5	6.9	5.5
	ResNet34	10.2	6.9	6.6	5.1
	ResNet50	8.4	9.9	7.6	5.0
	ResNet101	8.5	9.8	6.9	5.3
	ResNet152	8.5	9.7	7.3	4.7
	VGG11	11.8	10.7	9.4	8.2
	VGG13	10.3	8.8	8.4	6.7
	VGG16	11.0	11.4	8.5	6.7
	VGG19	12.1		8.6	6.9
ImageNet	ResNet18	41.1	33.1	33.5	31.5
	ResNet34	39.9	30.6	30.1	27.2
	ResNet50	34.8	31.8	28.2	25.0
	ResNet101	32.9	29.9	26.9	22.9
	ResNet152	31.9	29.1	27.6	22.6

(a) VGG16

(b) VGG16 +wd

Figure 11: **Whole-layer robustness analysis with** VGG**16 on CIFAR10.** The subfigures show how training with weight decay (+wd) affects the whole-layer robustness patterns.

(a) ResNet50

(b) ResNet50 +wd
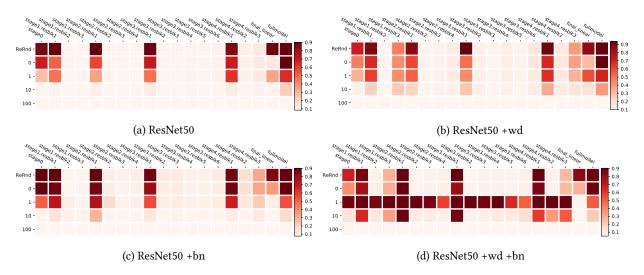
(c) ResNet50 +bn

(d) ResNet50 +wd +bn

Figure 12: **Whole-layer robustness analysis with** ResNet**50 on CIFAR10.** The subfigures show how training with weight decay (+wd) and batch normalization (+bn) affects the whole-layer robustness patterns.

(a) ResNet50

(b) ResNet50 +wd

(c) ResNet50 +bn
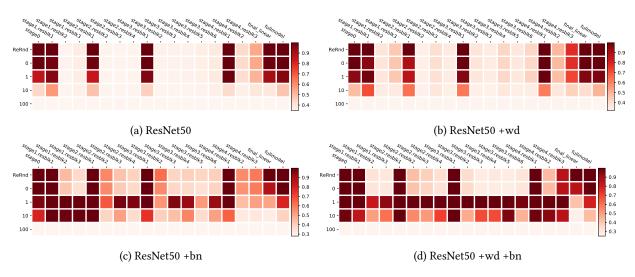
(d) ResNet50 +wd +bn

Figure 13: **Whole-layer robustness analysis with** ResNet**50 on ImageNet.** The subfigures show how training with weight decay (+wd) and batch normalization (+bn) affects the whole-layer robustness patterns.
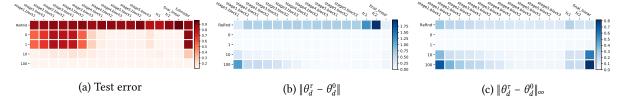
(a) Test error   (b) $\|\theta_d^\tau - \theta_d^0\|$ (c) $\|\theta_d^\tau - \theta_d^0\|_\infty$

Figure 14: **Whole-layer robustness studies of** VGG**16 on CIFAR10.** (a) shows the robustness analysis measured by the test error rate. (b) shows the normalized ℓ_2 distance of the parameters at each layer to the version realized during the re-randomization and re-initialization analysis. (c) is the same as (b), except with the ℓ_∞ distance.
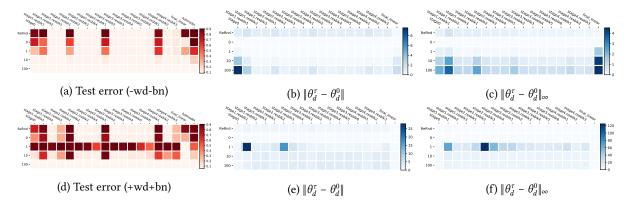

(a) Test error (-wd-bn)   (b) $\|\theta_d^\tau - \theta_d^0\|$ (c) $\|\theta_d^\tau - \theta_d^0\|_\infty$

(d) Test error (+wd+bn)   (e) $\|\theta_d^\tau - \theta_d^0\|$ (f) $\|\theta_d^\tau - \theta_d^0\|_\infty$

Figure 15: **Layer robustness for** ResNet**50 on CIFAR10.** Layouts are the same as in Fig. 14. The first row (a-c) is for ResNet50 trained without weight decay and batch normalization. The second row (d-f) is with weight decay and batch normalization.

two distances (the normalized 2 and ∞ norms) we measured. In this appendix, we list results on other models and datasets studied in this paper for comparison.

Fig. 14 shows the whole-layer robustness plot along with the layer-wise distance plots for VGG16 trained on CIFAR10. We found that the ℓ_∞ distance of the top layers are large, but the model is robust when we re-initialize those layers. However, the normalized ℓ_2 distance seem to be correlated with the whole-layer robustness patterns: the upper layers that are more robust moved smaller distances from their initialized values during training.

Similar plots for ResNet50 on CIFAR10 and ImageNet are shown in Fig. 15 and Fig. 16, respectively. In each of the figures, we also show extra results for models trained with weight decay and batch normalization. For the case without weight decay and batch normalization, we can see a weak correlation: the layers that are critical have slightly larger distances to their random initialization values. For the case with weight decay and batch normalization, the situation is less clear. First of all, in Fig. 15(e-f), we see very large distances in a few layers at checkpoint-1. This provides a potential explanation to the mysterious pattern that re-initialization to checkpoint-1 is more sensitive than to checkpoint-0. Similar observations can be found in Fig. 16(e-f) for ImageNet.

E   Alternative visualizations

The empirical results on layer robustness are mainly visualized as heatmaps in the main text. The heatmaps allow uncluttered comparison of the results across layers and training epochs. However, it is not easy to tell the difference between numerical values that are close to each other from the color coding. In this section, we provide alternative visualizations that shows the same results with line plots. In particular, Fig. 17 shows the whole-layer robustness analysis for VGG16 on CIFAR10. Fig. 18 shows the results for ResNet50 on CIFAR10 and ImageNet, respectively.
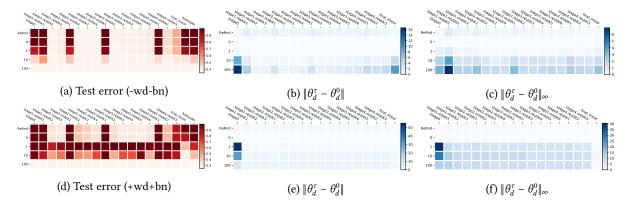
(a) Test error (-wd-bn)

(b) $\|\theta_d^\tau - \theta_d^0\|$

(c) $\|\theta_d^\tau - \theta_d^0\|_\infty$

(d) Test error (+wd+bn)

(e) $\|\theta_d^\tau - \theta_d^0\|$

(f) $\|\theta_d^\tau - \theta_d^0\|_\infty$

Figure 16: **Whole-layer robustness studies of** ResNet**50 on ImageNet.** Layouts are the same as in Fig. 14. The first row (a-c) is for ResNet50 trained without weight decay and batch normalization. The second row (d-f) is with weight decay and batch normalization.

(a) Test error

(b) $\|\theta_d^\tau - \theta_d^0\|$
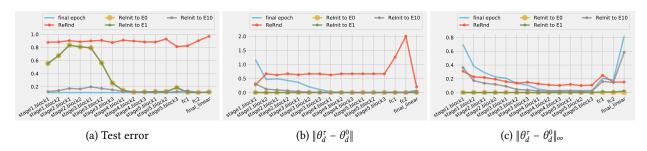
(c) $\|\theta_d^\tau - \theta_d^0\|_\infty$

Figure 17: **Alternative visualization of layer robustness analysis for** VGG**16 models on CIFAR10**. This shows the same results as Fig. 14, but shown as curves instead of heatmaps.
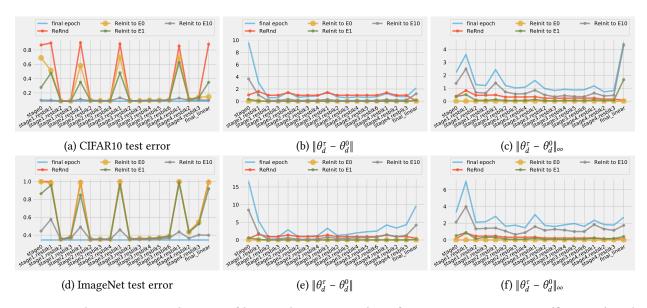

(a) CIFAR10 test error

(b) $\|\theta_d^\tau - \theta_d^0\|$

(c) $\|\theta_d^\tau - \theta_d^0\|_\infty$

(d) ImageNet test error

(e) $\|\theta_d^\tau - \theta_d^0\|$

(f) $\|\theta_d^\tau - \theta_d^0\|_\infty$

Figure 18: **Alternative visualization of layer robustness analysis for** ResNet**50 on CIFAR10 (first row) and ImageNet (second row)**.