SENSITIVITY AND GENERALIZATION IN NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

In practice it is often found that large over-parameterized neural networks generalize better than their smaller counterparts, an observation that appears to conflict with classical notions of function complexity, which typically favor smaller models. In this work, we investigate this tension between complexity and generalization through an extensive empirical exploration of two natural metrics of complexity related to sensitivity to input perturbations. Our experiments survey thousands of models with different architectures, optimizers, and other hyper-parameters, as well as four different image classification datasets.

We find that trained neural networks are more robust to input perturbations in the vicinity of the training data manifold, as measured by the input-output Jacobian of the network, and that this correlates well with generalization. We further establish that factors associated with poor generalization – such as full-batch training or using random labels – correspond to higher sensitivity, while factors associated with good generalization – such as data augmentation and ReLU non-linearities – give rise to more robust functions. Finally, we demonstrate how the input-output Jacobian norm can be predictive of generalization at the level of individual test points.

1 INTRODUCTION

The empirical success of deep learning eludes interpretation through lenses of computational complexity (Blum & Rivest, 1988), numerical optimization (Choromanska et al., 2015; Goodfellow & Vinyals, 2014; Dauphin et al., 2014) and classical statistical learning theory (Zhang et al., 2016): neural networks are highly non-convex models with extreme capacity that train fast and generalize well. In fact, not only do large networks demonstrate good test performance, but *larger* networks often generalize *better*. This phenomenon has been observed in targeted experiments (Neyshabur et al., 2015), historical trends of Deep Learning competitions (Canziani et al., 2016), and in the course of this work (Figure 1).

This observation is at odds with Occam's razor, the principle of parsimony, as applied to the intuitive notion of function complexity (see §A.1 for extended discussion). To resolve the apparent contradiction we examine complexity of functions in conjunction with the input domain. $f(x) = x^3 \sin(x)$ may seem decisively more complex than $g(x) = x$. But restrained to a narrow input domain of $[-0.01, 0.01]$ they appear differently: g remains a linear function of the input, while $f(x) = \mathcal{O}\left(x^4\right)$ resembles a constant 0. In this work we find that such intuition applies to neural networks, that behave very differently close to the data manifold than away from it (§4.1).

We therefore analyze the complexity of models through their capacity to distinguish different inputs in the neighborhood of datapoints, or, in other words, their sensitivity. We study two simple metrics presented in §3 and find that one of them, norm of the input-output Jacobian, correlates with generalization in a very wide variety of scenarios (§4.2, §4.3), up to being predictive of generalization at individual test points (§4.4).

This work considers sensitivity only in the context of image classification tasks. We interpret the observed correlation with generalization as an expression of a universal prior on (natural) image classification functions that favor robustness (see §A.1 for details). Care should be taken when extrapolating our findings to other tasks, where such prior may not be justified.

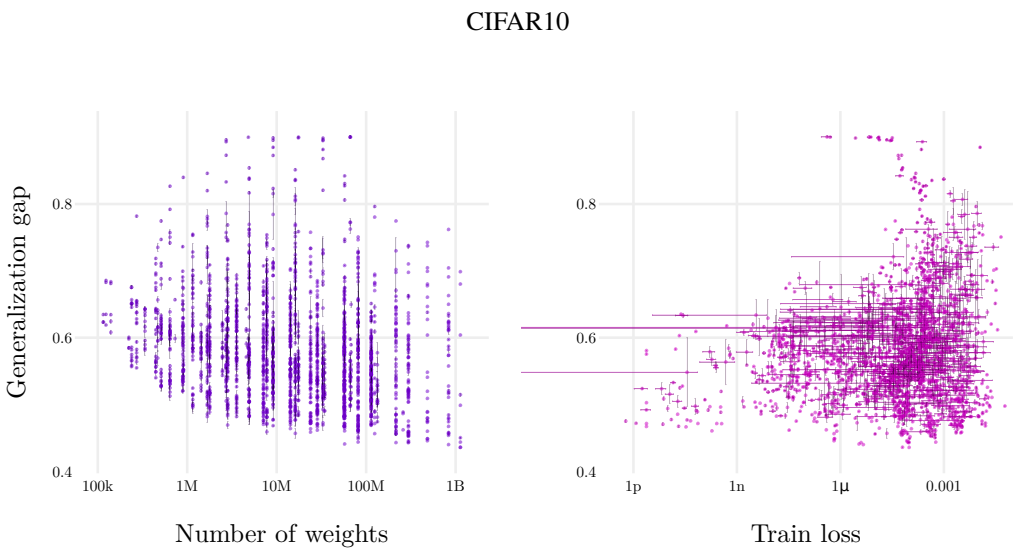

Figure 1: 2160 networks trained to 100% training accuracy on CIFAR10 (see §A.4.5 for experimental details). **Left**: while increasing capacity of the model allows for overfitting (top), very few models do, and a model with the maximum parameter count yields the best generalization (bottom right). **Right**: train loss does not correlate with generalization, and the best model (minimum along the y-axis) has training loss many orders of magnitude higher than models that generalize worse (left). This observation rules out underfitting as the reason for poor generalization in low-capacity models. See (Neyshabur et al., 2015) for similar findings in the case of achievable 0 training loss.

2 Related Work and Summary of Contributions

We analyze complexity of neural networks through (1) estimating the number of linear regions a network splits the input space into, and (2) measuring the norm of the input-output Jacobian within such regions.

A few prior works have examined measures related to the ones we consider. In particular, Pascanu et al. (2013); Montúfar et al. (2014); Raghu et al. (2016) have investigated the expressive power of fully-connected neural networks built out of piecewise-linear activation functions. Such functions are themselves piecewise-linear over their input domain, so that the number of linear regions into which input space is divided is a measure of how nonlinear the function is. A function with many linear regions has the capacity to build complex, flexible decision boundaries. It was argued in (Pascanu et al., 2013; Montúfar et al., 2014) that an upper bound to the number of linear regions scales exponentially with depth but polynomially in width, for specific constructions. Raghu et al. (2016) derived a tight analytic bound, and considered the number of linear regions for generic networks with random weights, as would be appropriate, for instance, at initialization. However, the evolution of this measure after training has not been investigated before. We examine a related measure, the number of hidden unit transitions along one-dimensional trajectories in input space, for trained networks. Further motivation for this measure will be discussed in §3.

Another perspective on function complexity can be gained by studying their robustness to perturbations to the input. Indeed, Rasmussen & Ghahramani (2000) demonstrate on a toy problem how complexity as measured by the number of parameters may be of limited utility for model selection, while measuring the output variation allows to invoke Occam's razor. In this work we apply related ideas to a large-scale practical context of neural networks with up to a billion free parameters (§4.2, §4.3), and discuss potential ways in which sensitivity permits the application of Occam's razor to neural networks (§A.1).

Sensitivity to inputs has attracted a lot of interest in the context of adversarial examples (Szegedy et al., 2013). Several attacks locate points of poor generalization in the directions of high sensitivity of the network (Goodfellow et al., 2014; Papernot et al., 2016; Moosavi-Dezfooli et al., 2016), while certain defences regularize the model by penalizing sensitivity (Gu & Rigazio, 2014) or employing

decaying (hence more robust) non-linearities (Kurakin et al., 2016). However, work on adversarial examples relates highly specific perturbations to a similarly specific kind of generalization (i.e. performance on a very small, adversarial subset of the data manifold), while this paper analyzes average-case sensitivity (§3) and typical generalization.

In the context of regularization, increasing robustness to perturbations is a widely-used strategy: data augmentation, noise injection (Jiang et al., 2009), weight decay (Krogh & Hertz, 1992), and max-pooling all indirectly reduce sensitivity of the model to perturbations, while Rifai et al. (2011); Sokolic et al. (2017) explicitly penalize the Frobenius norm of the Jacobian in the training objective. Sokolic et al. (2017) also provide theoretical support for the relevance of the input-output Jacobian to generalization. They derive bounds for the generalization gap, in terms of the Jacobian norm, within the framework of algorithmic robustness (Xu & Mannor, 2012), wherein an algorithm is robust if, whenever a testing sample is close to a training sample, the test loss is close to the training loss. Our results provide empirical support for their conclusions through an extensive number of experiments.

In this work we relate several of the mentioned above regularizing techniques to sensitivity, demonstrating through extensive experiments that improved generalization is consistently coupled with better robustness as measured by a single metric, the input-output Jacobian norm (§4.2). While some of these findings confirm common-sense expectations (data augmentation reduces sensitivity, Figure 6), others challenge our intuition of what makes a neural network robust (ReLU-networks, with unbounded activations, are more robust than saturating HardSigmoid-networks, Figure 7).

One such finding demonstrates an inductive bias towards robustness in stochastic mini-batch optimization compared to full-batch training (Figure 8). Interpreting this regularizing effect in terms of sensitivity is not new (Hochreiter & Schmidhuber, 1997; Keskar et al., 2016), yet we provide a novel perspective by relating it to reduced sensitivity to inputs instead of parameters.

The inductive bias of SGD ("implicit regularization") has been previously studied in (Neyshabur et al., 2015), where it was shown through rigorous experiments how increasing the width of a single-hidden-layer network improves generalization, and an analogy with matrix factorization was drawn to motivate constraining the norm of the weights instead of their count. Neyshabur et al. (2017) further explored several weight-norm based measures of complexity that do not scale with the size of the model. One of our measures, the Frobenius norm of the Jacobian is of similar nature (since the Jacobian matrix size is determined by the task and not by a particular network architecture). However, this particular metric was not considered, and, to the best of our knowledge, we are the first to evaluate it in a large-scale setting (e.g. our networks are up to 65 layers deep and up to 2^{16} units wide).

Finally, to the best of our knowledge, we are the first to compare behavior of a trained network on and off the data manifold (§4.1). The contrast between the two (Figure 2) further strengthens the empirical connection we draw between sensitivity and generalization; it also confirms that, as mentioned in §1, if a certain quality of a function is to be used for model comparison, input domain should always be accounted for.

3 Sensitivity Metrics

We propose two simple measures of sensitivity for a neural network $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^n$ with respect to its input $\mathbf{x} \in \mathbb{R}^d$ (the output being unnormalized logits of the n classes). Assume $\mathbf{f}$ employs a piecewise-linear activation function, like ReLU. Then $\mathbf{f}$ itself, as a composition of linear and piecewise-linear functions, is a piecewise-linear map, splitting the input space $\mathbb{R}^d$ into disjoint regions, implementing a single linear mapping on each. Then we can measure two aspects of sensitivity by answering

1. How does the output of the network change as the input is perturbed within the linear region?
2. How likely is the linear region to change in response to change in the input?

We quantify these qualities as follows:

1. For a local sensitivity measure we adopt the Frobenius norm of the class probabilities Jacobian $\mathbf{J}(\mathbf{x}) = \partial \mathbf{f}_\sigma(\mathbf{x}) / \partial \mathbf{x}^\mathbf{T}$ (with $J_{ij}(\mathbf{x}) = \partial \left[\mathbf{f}_\sigma(\mathbf{x})\right]_i / \partial x_j$), where $\mathbf{f}_\sigma = \boldsymbol{\sigma} \circ \mathbf{f}$ with $\boldsymbol{\sigma}$

being the softmax function. Given points of interest $\mathbf{x}_{\text{test}}$, we estimate the sensitivity of the function in those regions with the average Jacobian norm:

$$\mathbb{E}_{\mathbf{x}_{\text{test}}}\left[\left\|\mathbf{J}\left(\mathbf{x}_{\text{test}}\right)\right\|_F\right],$$

that we will further refer to as simply "Jacobian norm". Note that this does not require the label of $\mathbf{x}_{\text{test}}$.

Interpretation. The Frobenius norm $\|\mathbf{J}(\mathbf{x})\|_F = \sqrt{\sum_{ij} J_{ij}(\mathbf{x})^2}$ estimates the average-case sensitivity of $\mathbf{f}_\sigma$ around $\mathbf{x}$. Indeed, consider an infinitesimal Gaussian perturbation $\Delta\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \epsilon\mathbf{I})$: the expected magnitude of the output change is then

$$\begin{aligned}\mathbb{E}_{\Delta\mathbf{x}}\left[\left\|\mathbf{f}_\sigma(\mathbf{x}) - \mathbf{f}_\sigma(\mathbf{x}+\Delta\mathbf{x})\right\|_2^2\right] &\approx \mathbb{E}_{\Delta\mathbf{x}}\left[\left\|\mathbf{J}(\mathbf{x})\Delta\mathbf{x}\right\|_2^2\right] = \mathbb{E}_{\Delta\mathbf{x}}\Big[\sum_i\Big(\sum_j J_{ij}x_j\Big)^2\Big] \\ &= \sum_{ijj'} J_{ij}J_{ij'}\mathbb{E}_{\Delta\mathbf{x}}\left[x_j x_{j'}\right] = \sum_{ij} J_{ij}^2\mathbb{E}_{\Delta\mathbf{x}}\left[x_j^2\right] \\ &= \epsilon\left\|\mathbf{J}(\mathbf{x})\right\|_F^2.\end{aligned}$$

2. To detect a change in linear region (further called a "transition"), we need to be able to identify it. We do this analogously to Raghu et al. (2016). For a network with piecewise-linear activations, we can, given an input $\mathbf{x}$, assign a code to each neuron in the network $\mathbf{f}$, that identifies the linear region of the pre-activation of that neuron. E.g. each ReLU unit will have 0 or 1 assigned to it if the pre-activation value is less or greater than 0 respectively. Similarly, a ReLU6 unit (see definition in §A.3) will have a code of 0, 1, or 2 assigned, since it has 3 linear regions[1]. Then, a concatenation of codes of all neurons (denoted by $\mathbf{c}(\mathbf{x})$) in the network uniquely identifies the linear region of the input $\mathbf{x}$[2].

   Given this encoding scheme, we can detect a transition by detecting a change in the code. We then sample k points on a closed one-dimensional trajectory $\mathcal{T}(\mathbf{x})$ in the input space passing through the point $\mathbf{x}$ and count transitions $t(\mathbf{x})$ along it to quantify the number of linear regions:

$$t(\mathbf{x}) \coloneqq \sum_{i=0}^{k-1}\left[\mathbf{c}\left(\mathbf{x}'_i\right) \neq \mathbf{c}\left(\mathbf{x}'_{(i+1)\%k}\right)\right] \approx \oint_{\mathbf{x}'\in\mathcal{T}(\mathbf{x})}\left\|\frac{\partial\mathbf{c}(\mathbf{x}')}{\partial\left(d\mathbf{x}'\right)}\right\|_1 d\mathbf{x}', \tag{1}$$

   where $[\cdot]$ is the Iverson bracket and the norm of the directional derivative $\|\partial\mathbf{c}(\mathbf{x}')/\partial\left(d\mathbf{x}'\right)\|_1$ amounts to a Dirac delta function at each transition. As the sampling discretization k goes to infinity, the evaluation of the integral becomes exact[3].

   By sampling multiple such trajectories around different points, we estimate the sensitivity metric:

$$\mathbb{E}_{\mathbf{x}_{\text{test}}}\left[t\left(\mathbf{x}_{\text{test}}\right)\right],$$

   that we will further refer to as simply "transitions" or "number of transitions."

   To assure the sampling trajectory is close to the data manifold (since this is the region of interest), we construct it through horizontal translations of the image $\mathbf{x}_{\text{test}}$ in pixel space (Figure 2, right). We similarly augment our training data with horizontal and vertical translations in the corresponding experiments (Figure 6). As earlier, this metric does not require knowing the label of $\mathbf{x}_{\text{test}}$.

   Interpretation. We can draw a qualitative parallel between the number of transitions and curvature of the function. $t(\mathbf{x})$ in Equation 1 amounts to the total ℓ_1-norm of the directional derivative of $\mathbf{c}(\mathbf{x})$ along $\mathcal{T}(\mathbf{x})$, which is $\mathbf{0}$ everywhere except for the transition boundaries.

[1] For a non-piecewise-linear activation like Tanh, we consider 0 as the boundary of two regions and find this metric qualitatively similar to counting transitions of a piecewise-linear non-linearity.

[2] Such encoding guarantees that different regions obtain different codes, but different codes might be assigned to the same region if all the neurons in any layer of the network are saturated (or if weights leading from the transitioning unit to active units are exactly zero, or exactly cancel). However, the probability of such an arrangement drops exponentially with width and hence is ignored in this work.

[3] Apart from a dense-enough sampling we rely on no two neurons transitioning simultaneously, which is extremely unlikely in the context of random initialization and stochastic optimization.

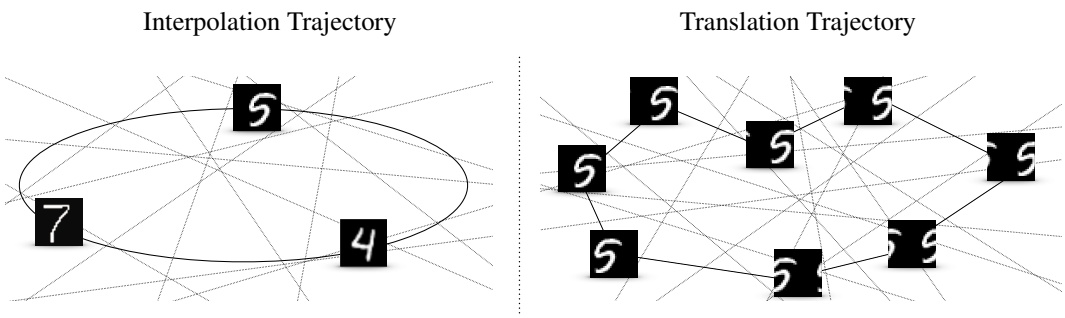

Figure 2: Two types of 1D trajectories in input space considered for transition counting. Boundaries between different linear regions are depicted with dotted lines[4]. **Left**: a circular interpolation between three training points results in samples between them lying outside of the data manifold, and as such allows comparison of the metrics of interest close to the data and away from it (see §4.1 for results and §A.4.2 for experimental details). **Right**: an interpolation between 28 horizontal translations of a single digit results in a complex trajectory that constrains all points to lie close to the translation-augmented data, and allows for a more realistic estimate of how often a model does transitions around the data. This metric is used to compare models in §4.2 and §4.3.

This property is shared by the derivative of $\mathbf{f}$, a constant input-logits Jacobian $\mathbf{J}_l$ within each linear region. Then, computing the total curvature of $\mathbf{f}$ along $\mathcal{T}(\mathbf{x})$ amounts to

$$\oint_{\mathbf{x}' \in \mathcal{T}(\mathbf{x})} \left\| \frac{\partial \mathbf{J}_l(\mathbf{x}')}{\partial (d\mathbf{x}')} \right\|_F d\mathbf{x}' \approx \frac{1}{2} \sum_{i=0}^{k-1} \left\| \mathbf{J}_l \left(\mathbf{x}'_{(i+1)\%k} \right) - \mathbf{J}_l \left(\mathbf{x}'_i \right) \right\|_F ,$$

where, as in Equation 1, the equality becomes exact for a sufficiently large k. This sum is similar to $t(\mathbf{x})$ as defined in Equation 1, but quantifies the amount of change in between two linear regions in a non-binary way. However, estimating this integral on a densely sampled trajectory is a computationally-intensive task, which is one reason we instead count transitions.

As such, on a qualitative level, the two metrics (Jacobian norm and number of transitions) track the first and second order terms of the Taylor expansion of the function.

Above we have defined two sensitivity metrics to describe the learned function around the data, on average. In §4.1 we analyze these measures on and off the data manifold by simply measuring them along circular trajectories in input space that intersect the data manifold at certain points, but generally lie away from it (Figure 2, left).

4 Experimental Results

We relate sensitivity and generalization through a sequence of experiments of increasing level of nuance.

In §4.1 we begin by comparing the sensitivity of trained neural networks on and off the training data manifold, i.e. in the regions of best and typical (over the whole input space) generalization.

In §4.2 we compare sensitivity of identical trained networks that differ in a single parameter which is important for generalization.

Further, §4.3 associates sensitivity and generalization in an unrestricted manner, i.e. comparing networks of a wide variety of hyper-parameters such as width, depth, non-linearity, weight initialization, optimizer, learning rate and batch size on CIFAR10 and CIFAR100, with a smaller, yet still substantial experiment confirming a similar trend on MNIST and Fashion-MNIST (Xiao et al., 2017).

[4]This depiction is only accurate in the case of a single-layer ReLU network with biases. Otherwise the partition into linear regions is more complex.

Finally, §4.4 explores how predictive sensitivity (as measured by the Jacobian norm) is for individual test points.

4.1 Sensitivity On and Off the Training Data Manifold

We analyze the behavior of a trained neural network near and away from training data. We do this by comparing sensitivity of the function along 3 types of trajectories:

1. A random ellipse. This trajectory is extremely unlikely to pass anywhere near the real data, and indicates how the function behaves in random locations of the input space that it never encountered during training.
2. An ellipse passing through 3 training points of different class (Figure 2, left). This trajectory does pass through the three data points, but in between it traverses images that are linear combinations of different-class images, and hence lie outside of the natural images space. Sensitivity of the function along this trajectory allows comparison of its behavior on and off the data manifold, as it approaches and moves away from the 3 anchor points.
3. An ellipse through 3 training points of the same class. This trajectory is similar to the previous one, but, given the dataset used in the experiment (MNIST), is expected to traverse overall closer to the data manifold, since linear combinations of the same digit are more likely to resemble a realistic image. Comparing transition density along this trajectory to the one through points of different classes allows further assessment of how sensitivity changes in response to approaching the data manifold.

We find that, according to both the Jacobian norm and transitions metrics, functions exhibit much more robust behavior around the training data (Figure 4). We further visualize this effect in 2D in Figure 3, where we plot the transition boundaries of the last (pre-logit) layer of a neural network before and after training. After training we observe that training points lie in regions of low transition density.

4.2 Sensitivity and Generalization Factors

In §4.1 we established that neural networks implement more robust functions in the vicinity of the training data manifold than away from it.

We now consider the more practical context of model selection. Given two perfectly trained neural networks, does the model with better generalization implement a less sensitive function?

We study approaches in the machine learning community that are commonly believed to influence generalization:

1. random labels (Figure 5);[5]
2. data augmentation (Figure 6);
3. ReLUs (Figure 7);
4. full-batch training (Figure 8).

We find that in each case, the change in generalization is coupled with the respective change in sensitivity (i.e. lower sensitivity corresponds to smaller generalization gap) as measured by the Jacobian norm (and almost always for the transitions metric).

[5]**How to read plots**: for many possible hyper-parameter configurations, we train two models that share all parameters and optimization procedure, but differ in a single binary parameter (i.e. trained on true or random labels; with or without data augmentation; etc). Out of all such network pairs, we select only those where each network reached 100% training accuracy on the whole training set (apart from the data augmentation study). The two generalization or sensitivity values are then used as the x and y coordinates of a point corresponding to this pair of networks. The position of the point with respect to the diagonal $y = x$ visually demonstrates which configuration has smaller generalization gap / lower sensitivity.

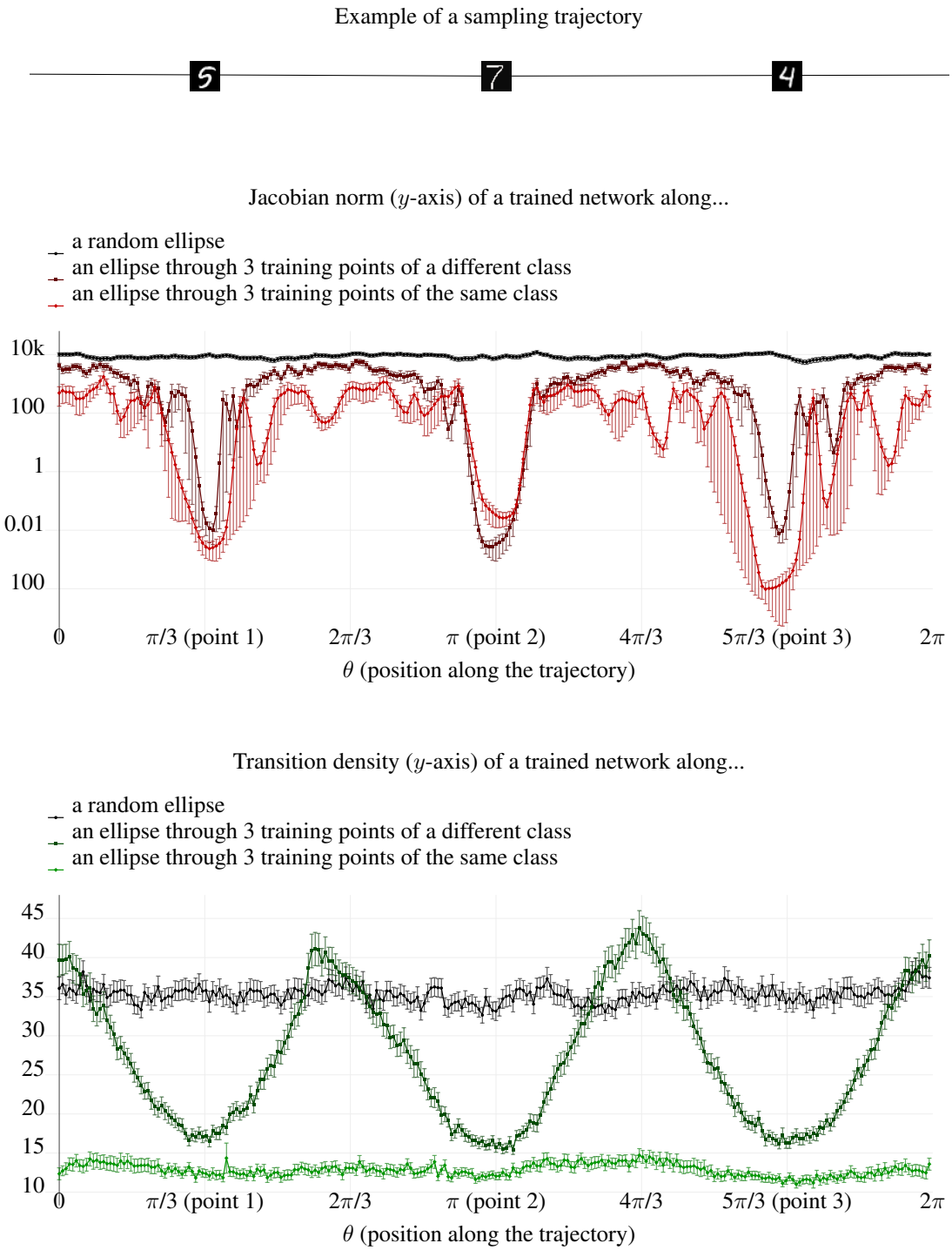

Figure 3: A 100%-accurate (on training data) MNIST network implements a function that is much more stable near training data than away from it. **Top**: depiction of a hypothetical circular trajectory (unrolled along the horizontal axis) passing through three digits of different classes, highlighting the training points locations ($\pi/3$, π, $5\pi/3$). **Center**: Jacobian norm as the input traverses an elliptical trajectory. Sensitivity drops significantly in the vicinity of training data while remaining uniform along random ellipses. **Bottom**: transition density behaves analogously. According to both metrics, as the input moves between points of different classes, the function becomes less stable than when it moves between points of the same class. This is consistent with the intuition that linear interpolations between different digits lie further from the data manifold than those between digits of the same class (which need not hold for more complex datasets). See §A.4.2 for experimental details.

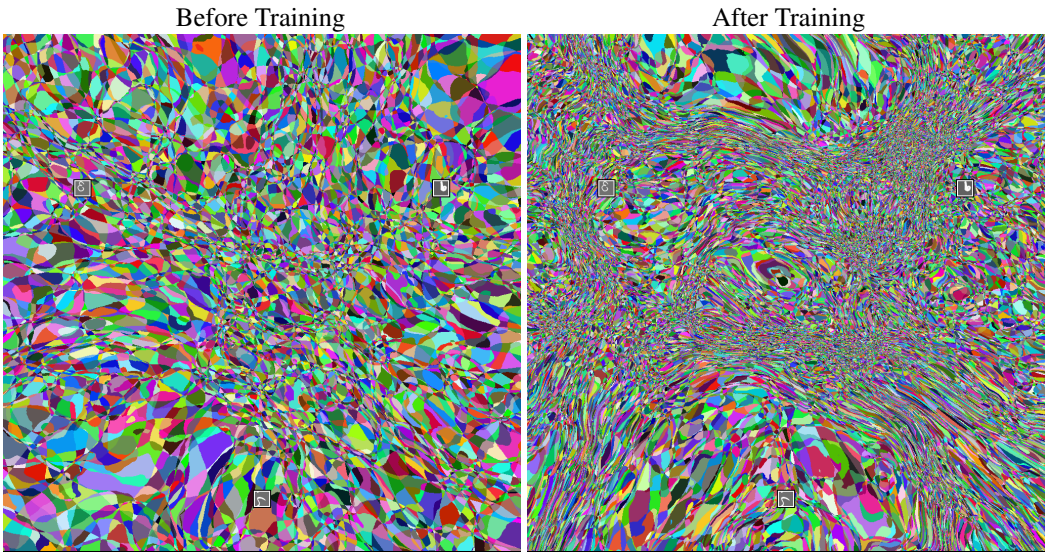

Figure 4: Transition boundaries of the last (pre-logits) layer over a 2-dimensional slice through the input space defined by 3 training points (indicated by inset squares). **Left**: boundaries before training. **Right**: after training, transition boundaries become highly non-isotropic, with training points lying in regions of lower transition density. See §A.4.3 for experimental details.

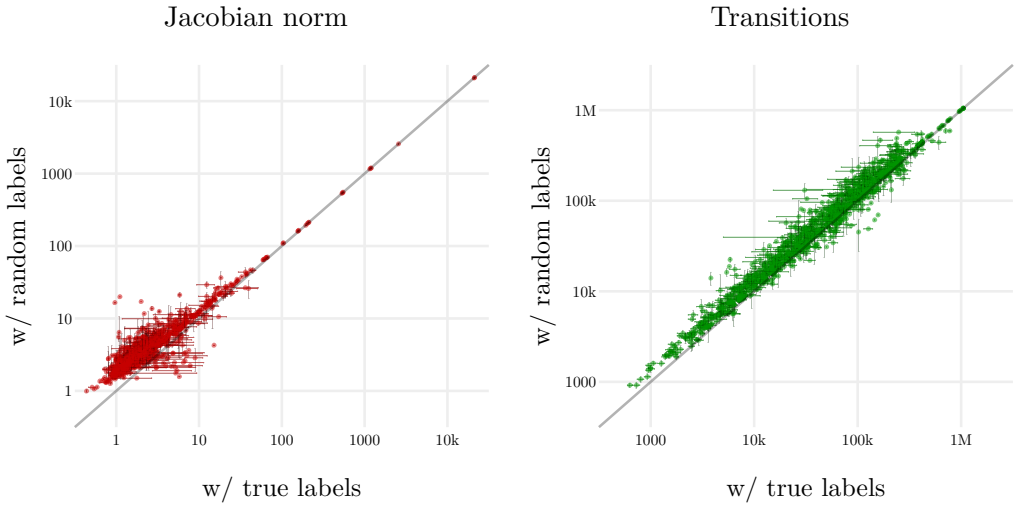

Figure 5: Networks trained on random labels consistently manifest higher sensitivity according to both transitions and Jacobian norm metrics (and, obviously, do not generalize). Each point on the plot corresponds to two neural networks that share all hyper-parameters and optimization procedure, but differ in training data labels as indicated by axes titles. The coordinates along each axis reflect the value of the quantity in the title of the plot in the respective setting (i. e. w/ true or random labels). All networks have reached 100% training accuracy on CIFAR10 in both settings. **Left**: Jacobian norm of networks that have fit random labels is significantly higher than that of networks fitting true labels. **Right**: networks trained on random labels make more transitions than those trained on true labels. See §A.4.5 for experimental details.

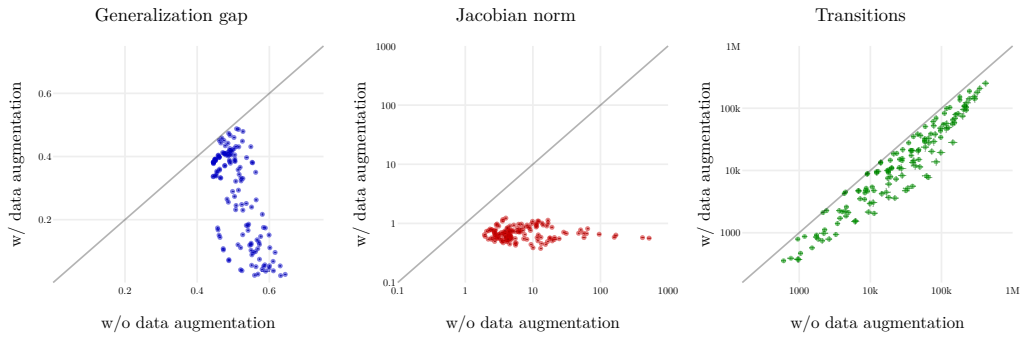

Figure 6: Data augmentation reduces the generalization gap and sensitivity according to both metrics. These plots are analogous to figures 5, 7 and 8, except that here networks are not filtered to have 100% train accuracy[6]. See §A.4.4 for experimental details.

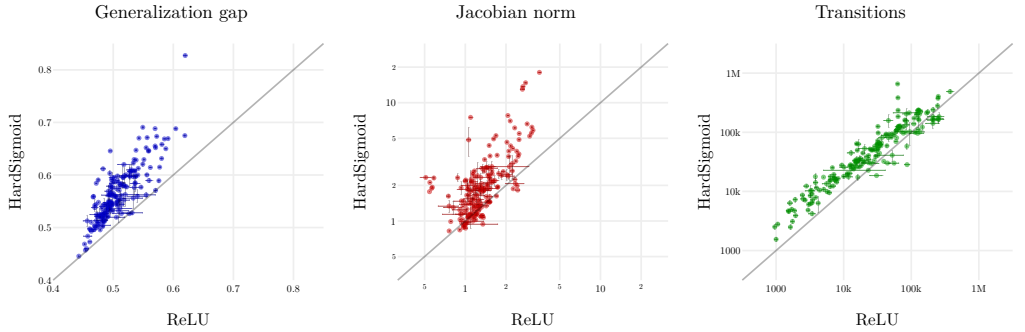

Figure 7: ReLU-networks generalize better than HardSigmoid-networks (see §A.3 for definition), and are less sensitive to inputs according to both Jacobian norm (despite HardSigmoid only having non-zero derivative on a small interval $[-0.5; 0.5]$) and transitions metrics. Each point corresponds to two 100%-accurate (on training set) neural networks different only in their activation function. See §A.4.5 for experimental details.

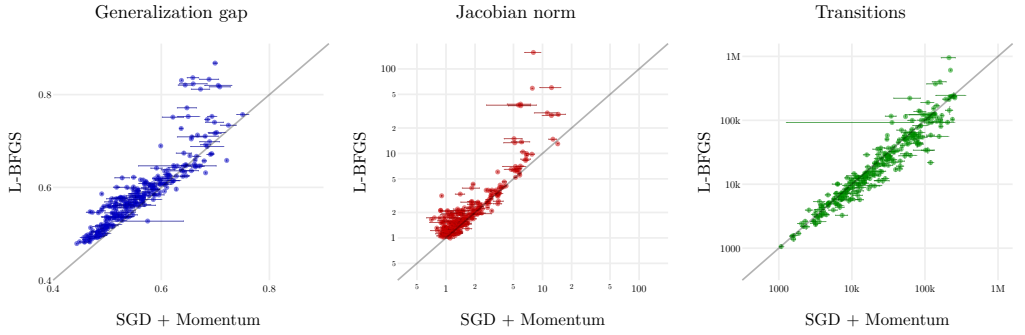

Figure 8: Full-batch training using L-BFGS generalizes worse than SGD with Momentum with batch of size 128, and, simultaneously, results in higher Jacobian norm. Each point corresponds to two 100%-accurate (on training set) neural networks different only in their optimizer and batch size. **Left**: L-BFGS generalizes worse than SGD + Momentum. **Center**: simultaneously, L-BFGS networks have higher Jacobian norm. **Right**: L-BFGS can lead to very high number of transitions (top), but for lower-capacity models it does not perform differently from models trained with SGD in terms of transitions. See §A.4.5 for experimental details.

[6]When applying data augmentation, the network is unlikely to encounter the canonical training data, hence few configurations achieved 100% training accuracy. However, we verified that all networks trained with data augmentation reached a higher test accuracy than their analogues without, ensuring that the generalization gap shrinks not simply because of lower training accuracy.

4.3 SENSITIVITY AND GENERALIZATION GAP

We now perform a large-scale experiment to establish direct relationships between sensitivity and generalization in a realistic setting. In contrast to §4.1, where we selected locations in the input space, and §4.2, where we varied a single binary parameter impacting generalization, we now sweep over many different architectural and optimization choices (§A.4.5).

Our main result is presented in Figure 9, indicating a strong relationship between the Jacobian norm and generalization. In contrast, Figure 10 demonstrates that the number of transitions is not alone sufficient to compare networks of different sizes, as the number of neurons in the networks has a strong influence on transition count.

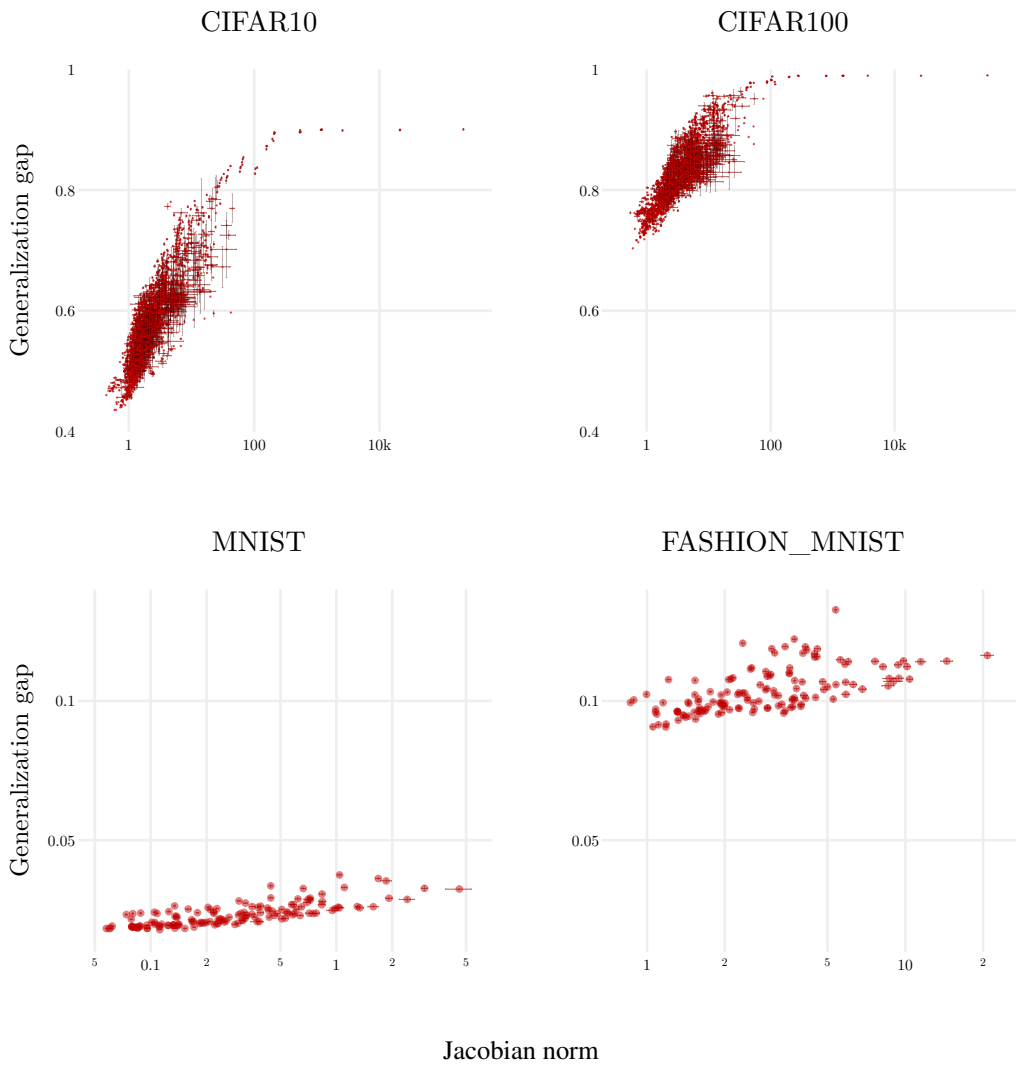

Figure 9: Jacobian norm correlates with generalization gap on all considered datasets. Each point corresponds to a network trained to 100% training accuracy (and at least 99.9% in the case of CIFAR100). See §A.4.4 and §A.4.5 for experimental details of bottom and top plots respectively.

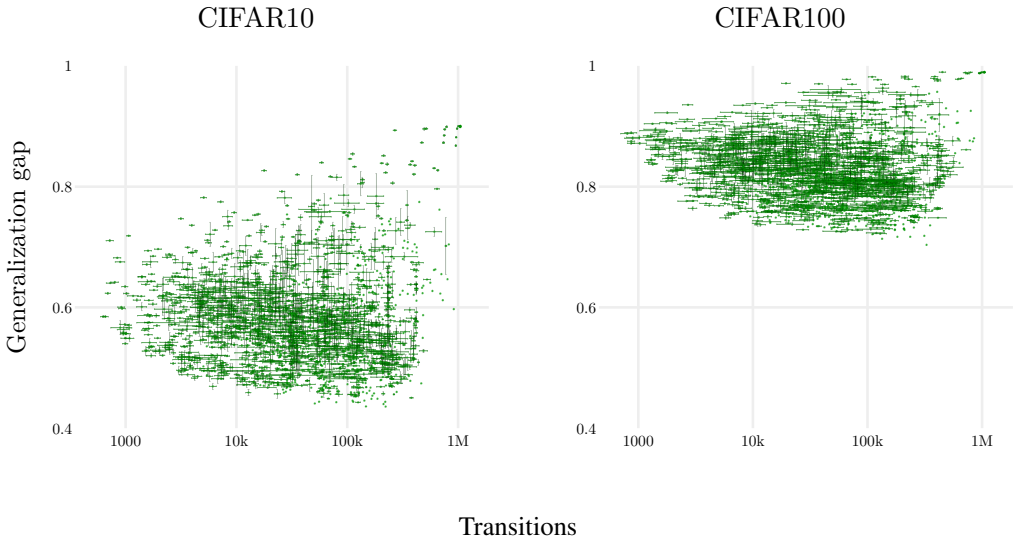

Figure 10: Number of transitions, in contrast to Figure 9, does not generally correlate with generalization gap. **Left**: 2160 networks with 100% train accuracy on CIFAR10. **Right**: 2097 networks with at least 99.9% training accuracy on CIFAR100. See §A.4.5 for experimental details

4.4 Sensitivity and Per-Point Generalization

In §4.3 we established a correlation between sensitivity (as measured by the Jacobian norm) and generalization averaged over a large test set (10^4 points). We now investigate whether the Jacobian norm can be predictive of generalization at individual points.

As demonstrated in Figure 11, Jacobian norm at a point is predictive of the cross-entropy loss, but the relationship is not a linear one, and not even bijective (see §A.2 for analytic expressions explaining it). In particular, certain misclassified points (right sides of the plots) have a Jacobian norm many orders of magnitude smaller than that of the correctly classified points (left sides). However, we do remark a consistent tendency for points having the highest values of the Jacobian norm to be mostly misclassified. This observation makes the jacobian norm a promising quantity to consider in the contexts of active learning and confidence estimation in future research.

5 Conclusion

We have investigated sensitivity of trained neural networks through the input-output Jacobian norm and linear regions counting in the context of image classification tasks. We have presented extensive experimental evidence indicating that the local geometry of the trained function as captured by the input-output Jacobian can be predictive of generalization in many different contexts, and that it varies drastically depending on how close to the training data manifold the function is evaluated. We further established a connection between the cross-entropy loss and the Jacobian norm, indicating that it can remain informative of generalization even at the level of individual test points.

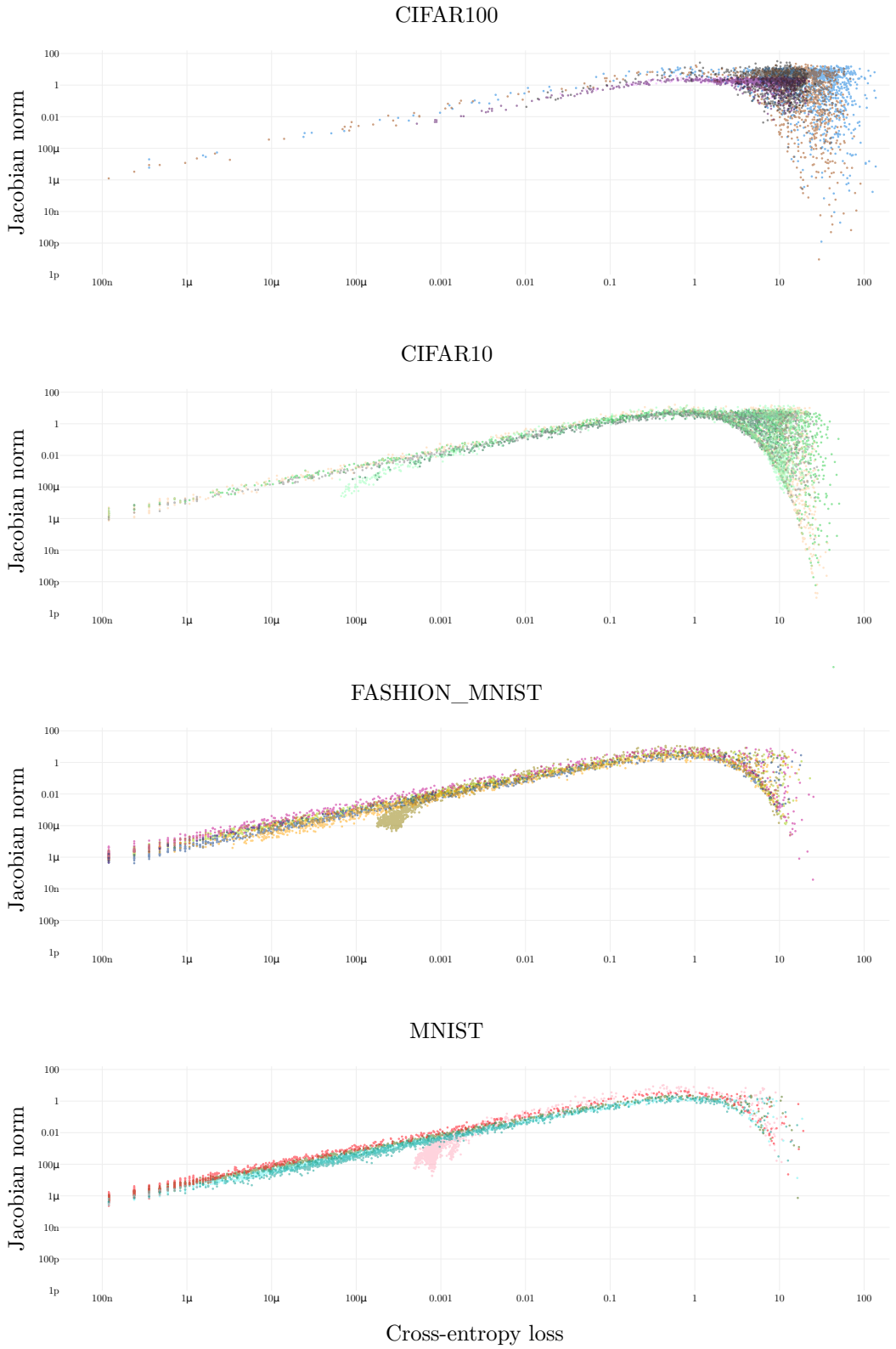

Figure 11: Jacobian norm plotted against individual test point loss. Each plot shows 5 random networks that fit the respective training set to a 100% with each network having a unique color. These plots experimentally confirm the relationship established in Figure 13 on different datasets and models. See §A.4.6 for experimental details.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Anonymous. Deep neural networks as gaussian processes. *International Conference on Machine Learning*, 2018.

Avrim Blum and Ronald L. Rivest. Training a 3-node neural network is np-complete. In *Machine Learning: From Theory to Applications*, 1988.

Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *CoRR*, abs/1605.07678, 2016.

Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pp. 192–204, 2015.

Yann Dauphin, Razvan Pascanu, aglar Gülehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *NIPS*, 2014.

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1487–1495. ACM, 2017.

Ian J. Goodfellow and Oriol Vinyals. Qualitatively characterizing neural network optimization problems. *CoRR*, abs/1412.6544, 2014.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Shixiang Gu and Luca Rigazio. Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068*, 2014.

Caglar Gulcehre, Marcin Moczulski, Misha Denil, and Yoshua Bengio. Noisy activation functions. In *International Conference on Machine Learning*, pp. 3059–3068, 2016.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning-lecture 6a-overview of mini-batch gradient descent, 2012.

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

William H Jefferys and James O Berger. Ockham's razor and bayesian analysis. *American Scientist*, 80(1):64–72, 1992.

Yulei Jiang, Richard M Zur, Lorenzo L Pesce, and Karen Drukker. A study of the effect of noise injection on the training of artificial neural networks. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pp. 1428–1432. IEEE, 2009.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky. Convolutional deep belief networks on cifar-10. 2010.

Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pp. 950–957, 1992.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

David J. C. MacKay. Bayesian methods for adaptive models. *PhD thesis, California Institute of Technology*, 1992.

David JC MacKay. Bayesian interpolation. 1991.

G. Montúfar, R. Pascanu, K. Cho, and Y. Bengio. On the Number of Linear Regions of Deep Neural Networks. *Neural Information Processing Systems*, February 2014.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. *arXiv preprint arXiv:1610.08401*, 2016.

Iain Murray and Zoubin Ghahramani. A note on the evidence and bayesian occam's razor, 2005.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Radford M. Neal. Priors for infinite networks (tech. rep. no. crg-tr-94-1). *University of Toronto*, 1994.

Radford M. Neal, Richard Mann, Carl Rasmussen, and Chris Williams. Bayesian learning for neural networks bayesian learning for neural networks. In *Lecture Notes in Statistics*, 1995.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. *Proceeding of the international Conference on Learning Representations workshop track*, abs/1412.6614, 2015.

Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. Exploring generalization in deep learning. *CoRR*, abs/1706.08947, 2017.

Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pp. 372–387. IEEE, 2016.

R. Pascanu, G. Montufar, and Y. Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations. *International Conference on Learning Representations*, December 2013.

B. Poole, S. Lahiri, M. Raghu, J. Sohl-Dickstein, and S. Ganguli. Exponential expressivity in deep neural networks through transient chaos. *Neural Information Processing Systems*, June 2016.

M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein. On the Expressive Power of Deep Neural Networks. *International Conference on Machine Learning*, June 2016.

Carl E. Rasmussen and Zoubin Ghahramani. Occam's razor. In *NIPS*, 2000.

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 833–840, 2011.

David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

Jure Sokolic, Raja Giryes, Guillermo Sapiro, and Miguel RD Rodrigues. Robust large margin deep neural networks. *IEEE Transactions on Signal Processing*, 2017.

Ercan Solak, Roderick Murray-Smith, William E Leithead, Douglas J Leith, and Carl E Rasmussen. Derivative observations in gaussian process models of dynamic systems. In *Advances in neural information processing systems*, pp. 1057–1064, 2003.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Matus Telgarsky. Representation benefits of deep feedforward networks. *CoRR*, abs/1509.08101, 2015.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Huan Xu and Shie Mannor. Robustness and generalization. *Machine learning*, 86(3):391–423, 2012.

C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *International Conference on Learning Representations*, November 2016.

A APPENDIX

A.1 DO NEURAL NETWORKS DEFY OCCAM'S RAZOR?

Here we briefly discuss the motivation of this work in the context of Occam's razor.

Occam's razor is an heuristic for comparing hypotheses based on their complexity, giving preference to simple models. It is commonly justified through the Bayesian "automatic Occam's razor" reasoning (Jefferys & Berger, 1992; MacKay, 1991) stating that given a dataset $\mathcal{D}$ and uniform prior on two competing hypotheses classes $\mathcal{H}_s$ (simple models, few parameters) and $\mathcal{H}_c$, (complex models, many parameters), the posterior $\mathbb{P}\left[\mathcal{H}|\mathcal{D}\right] \propto \mathbb{P}\left[\mathcal{D}|\mathcal{H}\right]$ is higher for a simple model $\mathcal{H}_s$ due to its likelihood $\mathbb{P}\left[\mathcal{D}|\mathcal{H}_s\right]$ having a smaller support in the dataset space (see Figure 12, left).

Occam's razor for neural networks. As seen in Figure 1, the above reasoning does not apply to neural networks: the best achieved generalization is obtained by a model that has around 10^4 more parameters than the simplest model capable of fitting the dataset (within the evaluated search space).

On one hand, (Murray & Ghahramani, 2005; Telgarsky, 2015) demonstrate on concrete examples that a high number of free parameters in the model doesn't necessarily entail high complexity. On the other hand, a large body of work on the expressivity of neural networks (Raghu et al., 2016; Poole et al., 2016; Montúfar et al., 2014) shows that their ability to compute complex functions increases rapidly with size, while (Zhang et al., 2016) validates that they also easily fit complex (even random) functions with stochastic optimization. This indicates that weights of a neural network may actually correspond to its usable capacity, and hence lead to "smearing" the likelihood $\mathbb{P}\left[\mathcal{D}|\mathcal{H}\right]$ along a very large space of datasets $\mathcal{D}'$, making the dataset of interest $\mathcal{D}$ less likely.

Potential issues. We conjecture two reasons for why Occam's razor may not apply:

1. The approximate computation involved in estimating the "Occam's factor" (MacKay, 1991; 1992) used to assess the shape of the likelihood curves in (Figure 12, left) involves assumptions that may not hold for neural network in the context of stochastic optimization, and, in particular, do not account for the combinatorial growth of the acceptable volume of parameter space as width increases. We conjecture that a realistic plot might look like the one shown in (Figure 12, right), which renders model comparison by their classes $\mathcal{H}$ impossible without highly precise knowledge of the likelihood curves.
2. For the proposed hypotheses classes (i.e. hypotheses binned by number of parameters) $\mathbb{P}\left[\mathcal{H}\right]$ is far from uniform. In fact, given the complexity of tasks we tackle with neural networks in practical applications, the prior might favor larger models to small ones (Neal et al., 1995).

We interpret this work as tackling the second issue: we suggest to define hypotheses classes based on sensitivity of the hypothesis (which yielded promising results in (Rasmussen & Ghahramani, 2000) on a toy task). Indeed, at least in the context of natural images classification, putting a prior on the number of parameters or Kolmogorov complexity of the hypothesis is extremely difficult. However, a statement that the true classification function is robust to small perturbations in the input is much easier to justify. As such, a prior $\mathbb{P}\left[\mathcal{H}\right]$ in favor of robustness over sensitivity might fare better than a prior on specific network hyper-parameters.

A.2 BOUNDING THE JACOBIAN NORM

Here we analyze the relationship between the Jacobian norm and the cross-entropy loss at individual test points.

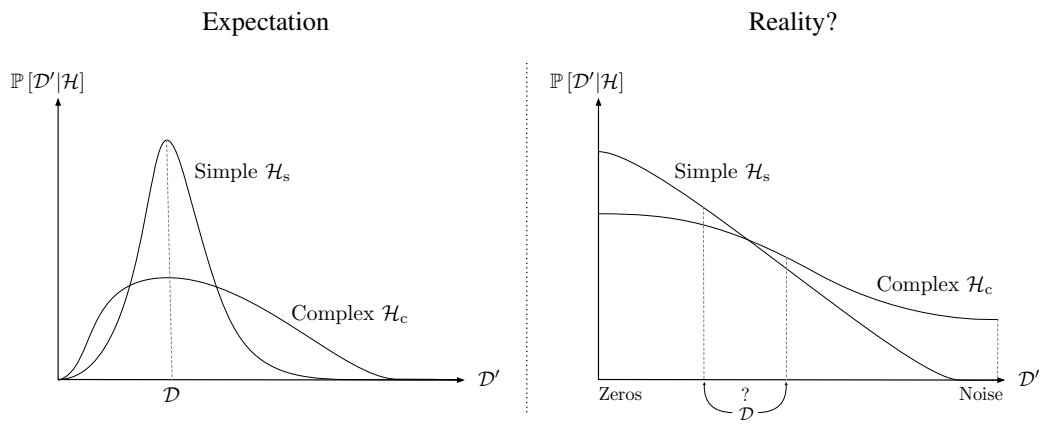

Figure 12: Occam's Razor: expectation vs hypothesized reality. **Left**: a classic depiction of Bayesian Occam's razor. Likelihood $\mathbb{P}[\mathcal{D}'|\mathcal{H}]$ of a simple model class $\mathcal{H}_s$ with few parameters has smaller support in the dataset space, hence is more peaked. If the class fits the dataset $\mathcal{D}$ well, it falls close to the peak and outperforms a complex class $\mathcal{H}_c$ with more parameters, having wider support. **Right**: suggested potential reality of neural networks. All datasets $\mathcal{D}'$ with inputs and targets dimensions matching those of a particular dataset $\mathcal{D}$ are sorted according to likelihood $\mathbb{P}[\mathcal{D}'|\mathcal{H}]$ of a complex class $\mathcal{H}_c$ from left to right along the horizontal axis. A simple class $\mathcal{H}_s$ peaks higher, but we don't know onto which side of the likelihood curves intersection $\mathcal{D}$ falls.

Target class Jacobian.We begin by relating the derivative of the target class probability $\mathbf{J}_{y(\mathbf{x})}$ to per-point cross-entropy loss $l(\mathbf{x}) = -\log\left[\mathbf{f}_\sigma(\mathbf{x})\right]_{y(\mathbf{x})}$ (where $y(\mathbf{x})$ is the correct integer class).

We will denote $\mathbf{f}_\sigma(\mathbf{x})$ by $\boldsymbol{\sigma}$ and drop the $\mathbf{x}$ argument to de-clutter notation (e.g. write $\mathbf{f}$ instead of $\mathbf{f}(\mathbf{x})$). Then the Jacobian can be expressed as

$$\mathbf{J} = \left[\left(\boldsymbol{\sigma}\mathbf{1}^T\right) \odot \left(\mathbf{I} - \boldsymbol{\sigma}\mathbf{1}^T\right)\right]\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}^T}\right),$$

where $\odot$ is the Hadamard element-wise product. Then indexing both sides of the equation at the correct class y yields

$$\mathbf{J}_y = \sigma_y\left(\left(\mathbf{e}_y - \boldsymbol{\sigma}\right)^T\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}^T}\right)\right),$$

where $\mathbf{e}_y$ is a vector of zeros everywhere except for $e_y = 1$. Taking the norm of both sides results in

$$\begin{aligned}\|\mathbf{J}_y\|_2^2 = \sigma_y^2 \sum_{k=1}^{d}\left[(1-\sigma_y)^2\left(\frac{\partial f_y}{\partial x_k}\right)^2 + \sum_{j\neq y}^{n}\left(\sigma_j\frac{\partial f_j}{\partial x_k}\right)^2\right] = \\ = \sigma_y^2\left[(1-\sigma_y)^2\sum_{k=1}^{d}\left(\frac{\partial f_y}{\partial x_k}\right)^2 + \sum_{j\neq y}^{n}\sigma_j^2\sum_{k=1}^{d}\left(\frac{\partial f_j}{\partial x_k}\right)^2\right] = \\ = \sigma_y^2\left[(1-\sigma_y)^2\left\|\frac{\partial f_y}{\partial \mathbf{x}^T}\right\|_2^2 + \sum_{j\neq y}^{n}\sigma_j^2\left\|\frac{\partial f_j}{\partial \mathbf{x}^T}\right\|_2^2\right] \quad (2)\end{aligned}$$

We now assume that magnitudes of the individual logit derivatives vary little in between logits and over the input space[7]:

$$\left\|\frac{\partial f_i}{\partial \mathbf{x}^T}\right\|_2^2 \approx \frac{1}{n}\mathbb{E}_{\mathbf{x}_{\text{test}}}\left\|\frac{\partial \mathbf{f}}{\partial \mathbf{x}_{\text{test}}^T}\right\|_F^2,$$

[7]In the limit of infinite width, and fully Bayesian training, deep network logits are distributed exactly according to a Gaussian process (Neal, 1994; Anonymous, 2018). Similarly, each entry in the logit Jacobian also corresponds to an independent draw from a Gaussian process (Solak et al., 2003). It is therefore plausible that the Jacobian norm, consisting of a sum over the square of independent Gaussian samples in the correct limits, will tend towards a constant.

which simplifies Equation 2 to

$$\|\mathbf{J}_y\|_2^2 \approx M\sigma_y^2 \left[(1-\sigma_y)^2 + \sum_{j \neq y}^{n} \sigma_j^2 \right],$$

where $M = \mathbb{E}_{\mathbf{x}_{\text{test}}} \left\| \partial \mathbf{f} / \partial \mathbf{x}_{\text{test}}^T \right\|_F^2 / n$. And since σ lies on the $(n-1)$-simplex Δ^{n-1}, we can bound:

$$\frac{(1-\sigma_y)^2}{n-1} \leqslant \sum_{j \neq y}^{n} \sigma_j^2 \leqslant (1-\sigma_y)^2,$$

and finally

$$\frac{n}{n-1} M\sigma_y^2 \left(1-\sigma_y\right)^2 \lessapprox \|\mathbf{J}_y\|_2^2 \lessapprox 2M\sigma_y^2 \left(1-\sigma_y\right)^2,$$

or, in terms of the cross-entropy loss $l = -\log \sigma_y$:

$$\sqrt{\frac{nM}{n-1}} \mathbb{e}^{-l} \left(1 - \mathbb{e}^{-l}\right) \lessapprox \|\mathbf{J}_y\|_2 \lessapprox \sqrt{2M} \mathbb{e}^{-l} \left(1 - \mathbb{e}^{-l}\right). \tag{3}$$

We validate these approximate bounds in Figure 13 (top).

Full Jacobian. Equation 3 establishes a close relationship between $\mathbf{J}_y$ and loss $l = -\log \sigma_y$, but of course, at test time we do not know the target class y. This allows us to only bound the full Jacobian norm from below:

$$\sqrt{\frac{nM}{n-1}} \mathbb{e}^{-l} \left(1 - \mathbb{e}^{-l}\right) \lessapprox \|\mathbf{J}_y\|_2 \leqslant \|\mathbf{J}\|_F. \tag{4}$$

For the upper bound, we assume the maximum-entropy case of σ_y: $\sigma_i \approx (1-\sigma_y)/(n-1)$, for $i \neq y$. The Jacobian norm is

$$\|\mathbf{J}\|_F^2 = \sum_{i=1}^{n} \|\mathbf{J}_i\|_2^2 = \|\mathbf{J}_y\|_2^2 + \sum_{i \neq y}^{n} \|\mathbf{J}_i\|_2^2,$$

where we the first summand becomes:

$$\|\mathbf{J}_y\|_2^2 \approx M\sigma_y^2 \left[\left(1-\sigma_y\right)^2 + (n-1) \left(\frac{1-\sigma_y}{n-1} \right)^2 \right] = \frac{Mn}{n-1} \sigma_y^2 \left(1-\sigma_y\right)^2,$$

and each of the others

$$\|\mathbf{J}_i\|_2^2 \approx M \left(\frac{1-\sigma_y}{n-1} \right)^2 \left[\left(1 - \frac{1-\sigma_y}{n-1} \right)^2 + \left(\sigma_y^2 + (n-2) \left(\frac{1-\sigma_y}{n-1} \right)^2 \right) \right] =$$

$$= \frac{M}{(n-1)^3} \left(1-\sigma_y\right)^2 \left(n\sigma_y^2 + n - 2\right)^2. \tag{5}$$

Adding $n-1$ of such summands to $\|\mathbf{J}_y\|_2^2$ results in

$$\|\mathbf{J}\|_F \approx \frac{\sqrt{M}}{(n-1)} (1-\sigma_y) \sqrt{n^2\sigma_y^2 + n - 2} = \frac{\sqrt{M}}{(n-1)} \left(1 - \mathbb{e}^{-l}\right) \sqrt{n^2 \mathbb{e}^{-2l} + n - 2}, \tag{6}$$

compared against the lower bound (Equation 4) and experimental data in figure 13.

A.3 Non-linearities Definitions

Following activation functions are used in this work:

1. ReLU (Nair & Hinton, 2010): $\max(x, 0)$;
2. ReLU6 (Krizhevsky, 2010): $\min\left(\max(x, 0), 6\right)$;
3. Tanh: hyperbolic tangent, $(e^x - e^{-x})/(e^x + e^{-x})$;
4. HardTanh (Gulcehre et al., 2016): $\min\left(\max(x, -1), 1\right)$;
5. HardSigmoid (Gulcehre et al., 2016): $\min\left(\max(x + 0.5, 0), 1\right)$;

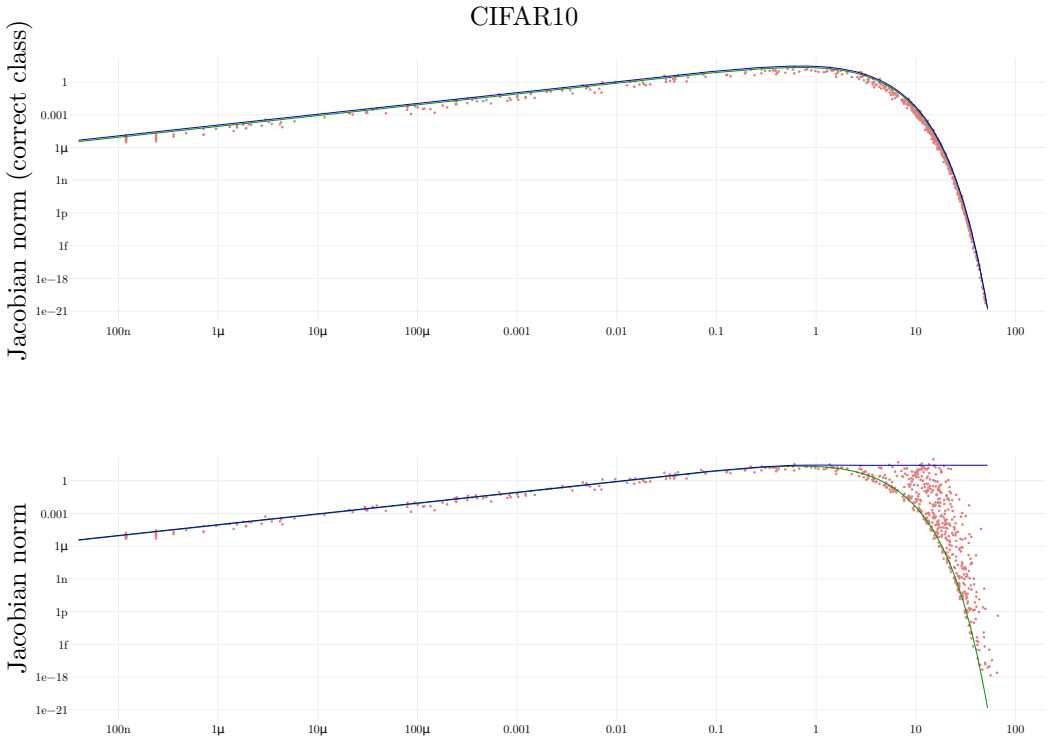

Figure 13: **Top**: Jacobian norm $\|\mathbf{J}_y(\mathbf{x})\|_2 = \left\|\partial \mathbf{f}_\sigma(\mathbf{x})_y / \partial \mathbf{x}^T\right\|_2$ of the true class y output probability is tightly related to the cross-entropy loss (x-axis). Each point corresponds to one of the 1000 test inputs to a 100% trained network on CIFAR10, while lines depict analytic bounds from Equation 3. **Bottom**: Same experiment plotting the full Jacobian norm $\|\mathbf{J}\|_F$ against cross-entropy. Solid lines correspond to the lower bound from Equation 4 and the norm approximation from Equation 6. See §A.4.6 for experimental details.

A.4 EXPERIMENTAL SETUP

All experiments were implemented in Tensorflow (Abadi et al., 2016) and executed with the help of Vizier (Golovin et al., 2017). All networks were trained with cross-entropy loss. All networks were trained without biases. All computations were done with 32-bit precision. Learning rate decayed by a factor of 0.1 every 500 epochs.

Unless specified otherwise, initial weights were drawn from a normal distribution with zero mean and variance $2/n$ for ReLU, ReLU6 and HardSigmoid; $1/n$ for Tanh and HardTanh, where n is the number of inputs to the current layer.

All inputs were normalized to have zero mean and unit variance, or, in other terms, lie on the d-dimensional sphere of radius $\sqrt{d}$, where d is the dimensionality of the input.

All reported values, when applicable, were evaluated on the whole training and test sets of sizes 50000 and 10000 respectively. E.g. "generalization gap" is defined as the difference between train and test accuracies evaluated on the whole train and test sets.

When applicable, all trajectories/surfaces in input space were sampled with 2^{20} points.

A.4.1 PLOTS AND ERROR BARS

All figures except for 11 and 13 are plotted with (pale) error bars (when applicable). The reported quantity was usually evaluated 8 times with random seeds from 1 to 8[8], unless specified otherwise. E.g. if a network is said to be 100%-accurate on the training set, it means that each of the 8 randomly-initialized networks is 100% accurate after training.

The error bar is centered at the mean value of the quantity and spans the standard error of the mean in each direction. If the bar appears to not be visible, it may be smaller than the mean value marker.

Weight initialization, training set shuffling, data augmentation, picking anchor points of data-fitted trajectories, selecting axes of a zero-centered elliptic trajectory depend on the random seed.

A.4.2 SENSITIVITY ALONG A TRAJECTORY

Relevant figure 3.

A 20-layer ReLU-network of width 200 was trained on MNIST 128 times, with plots displaying the averaged values.

A random zero-centered ellipse was obtained by generating two axis vectors with normally-distributed entries of zero mean and unit variance (as such making points on the trajectory have an expected norm equal to that of training data) and sampling points on the ellipse with given axes.

A random data-fitted ellipse was generated by projecting three arbitrary input points onto a plane where they fall into vertices of an equilateral triangle, and then projecting their circumcircle back into the input space.

A.4.3 LINEAR REGION BOUNDARIES

Relevant figure 4.

A 15-layer ReLU6-network of width 300 was trained on MNIST for 2^{18} steps using SGD with momentum (Rumelhart et al., 1988); images were randomly translated with wrapping by up to 4 pixels in each direction, horizontally and vertically, as well as randomly flipped along each axis, and randomly rotated by 90 degrees clockwise and counter-clockwise.

The sampling grid in input space was obtain by projecting three arbitrary input points into a plane as described in §A.4.2 such that the resulting triangle was centered at 0 and it's vertices were at a distance 0.8 form the origin. Then, a sampling grid of points in the $[-1; 1]^{\times 2}$ square was projected back into the input space.

[8] If a particular random seed did not finish, it was not taken into account; we believe this nuance did not influence the conclusions of this paper.

A.4.4 SMALL EXPERIMENT

Relevant figures: 6 and 9 (bottom).

All networks were trained for 2^{18} steps of batch size of 256 using SGD with momentum. Learning rate was set to 0.005 and momentum term coefficient to 0.9.

Data augmentation consisted of random translation of the input by up to 4 pixels in each direction with wrapping, horizontally and vertically. The input was also flipped horizontally with probability 0.5.

For each dataset, networks of width $\{100, 200, 500, 1000, 2000, 3000\}$, depth $\{2, 3, 5, 10, 15, 20\}$ and activation function $\{$ReLU, ReLU6, HardTanh, HardSigmoid$\}$ were evaluated 8 times each.

A.4.5 LARGE EXPERIMENT

Relevant figures: 1, 5, 7, 8, 9 (top).

335671 networks were trained for 2^{19} steps with random hyper-parameters; if training did not complete, a checkpoint at step 2^{18} was used instead, if available. When using L-BFGS, the maximum number of iterations was set to 2684. The space of available hyper-parameters included[9]:

1. CIFAR10 and CIFAR100 datasets cropped to a 24×24 center region;
2. all 5 non-linearities from §A.3;
3. SGD, Momentum, ADAM (Kingma & Ba, 2014), RMSProp (Hinton et al., 2012) and L-BFGS optimizers;
4. learning rates from $\{0.01, 0.005, 0.0005\}$, when applicable. Secondary coefficients were fixed at default values of Tensorflow implementations of respective optimizers;
5. batch sizes of $\{128, 512\}$ (unless using L-BFGS with the full batch of 50000);
6. standard deviations of initial weights from $\{0.5, 1, 4, 8\}$ multiplied by the default value described in §A.4;
7. widths from $\left\{1, 2, 4, \cdots, 2^{16}\right\}$;
8. depths from $\left\{2, 3, 5, \cdots, 2^{6} + 1\right\}$;
9. true and random training labels;
10. random seeds from 1 to 8.

A.4.6 PER-POINT GENERALIZATION

Relevant figures: 11 and 13.

Here we trained for 2^{16} steps and evaluated networks on random train and test subsets of sizes 5000[10] and 1000 respectively of CIFAR100, CIFAR10, Fashion-MNIST and MNIST. The hyper-parameters consisted of non-linearity (all functions from §A.3), width (50, 100 or 200) and depth (2, 5, 10, 20). Only one random seed (1) was considered. For each dataset, a random subset of 5 configurations among all the 100%-accurate (on training) networks was plotted.

Figure 13 was generated from a similar experiment, with the exception that the whole training set was used and 2^{18} training steps were done. A single random 100%-accurate (on training data) network was drawn to compare experimental measurements with analytic bounds on the Jacobian norm.

[9] Due to time and compute limitations, this experiment was set up such that configurations of small size were more likely to get evaluated (e.g. only a few networks of width 2^{16} were trained, and all of them had depth 2). However, based on our experience with smaller experiments (where each configuration got evaluated), we believe this did not bias the findings of this paper, while allowing them to be validated across a very wide range of scenarios.

[10] In a different experiment, precisely the one used in Figure 13, we trained on the whole training set of size 50000 and obtained qualitatively similar plots.