Are All Linear Regions Created Equal?

Matteo Gamba **Adrian Chmielewski-Anders** **Josephine Sullivan** **Hossein Azizpour**
Mårten Björkman
KTH Royal Institute of Technology, Stockholm, Sweden

Abstract

The number of linear regions has been studied as a proxy of complexity for ReLU networks. However, the empirical success of network compression techniques like pruning and knowledge distillation, suggest that in the overparameterized setting, linear regions density might fail to capture the effective nonlinearity. In this work, we propose an efficient algorithm for discovering linear regions and use it to investigate the effectiveness of density in capturing the nonlinearity of trained VGGs and ResNets on CIFAR-10 and CIFAR-100. We contrast the results with a more principled nonlinearity measure based on function variation, highlighting the shortcomings of linear regions density. Furthermore, interestingly, our measure of nonlinearity clearly correlates with model-wise deep double descent, connecting reduced test error with reduced nonlinearity, and increased local similarity of linear regions.

1 INTRODUCTION

Estimating the complexity of deep networks trained in practice is an open research problem posing several challenges (Kawaguchi et al., 2017; Neyshabur et al., 2015). For a network equipped with piece-wise linear activation functions – most prominently ReLU – one avenue for studying complexity is through the lens of *linear regions*, namely the connected components induced on the input space by the piece-wise affine function parameterized by the network (Balestriero et al., 2018; Montufar et al., 2014; Pascanu et al., 2013).

Early works on linear regions highlighted theoretical gains in model expressivity for deep networks, as opposed to wider and shallower ones (Cohen et al., 2016; Telgarsky, 2016; Håstad, 1986). Later studies mainly focused on estimating the *density* of linear regions, *i.e.* bounding and counting the number of affine components realized by a given network architecture, proposed as a measure for studying the complexity of hierarchical representations (Xiong et al., 2020; Hanin & Rolnick, 2019b; Novak et al., 2018; Serra et al., 2018; Arora et al., 2018; Montufar et al., 2014; Pascanu et al., 2013). Intuitively, the main rationale motivating such studies is that, in order for a network to model complex non-linear behaviour, the resulting piece-wise affine function should count many affine components.

Currently, it is debated whether linear region density is the most suited metric for capturing the complexity of piece-wise affine functions parameterized by deep networks (Trimmel et al., 2021; LeJeune et al., 2019; Hanin & Rolnick, 2019a), but there is no systematic study presenting evidence of where density fails.

Crucially, existing empirical studies of linear regions are limited to small networks, and have thus not explored the relationship between model size – which directly controls expressivity – and nonlinearity. In fact, existing numerical methods for density estimation (Novak et al., 2018) suffer from the limitations of uniform sampling (Figure 1, top), while exact analytic approaches are unable to scale to large networks (Zhang & Wu, 2020; Hanin & Rolnick, 2019a,b).

In this work, we systematically investigate whether all linear regions equally contribute nonlinearity relevant to learning for ReLU networks. We expose shortcomings of estimating the complexity of piece-wise affine functions exclusively by the number of their affine components, by contrasting linear region density to a more principled way of estimating nonlinearity. Intriguingly, using our nonlinearity measure, we empirically observe that for increasing model size, nonlinearity increases for overfitting networks and then decreases for large, generalizing ones (Figure 2); in line with the recently observed deep double descent phenomenon (Nakkiran et al., 2019; Belkin et al., 2019).

Proceedings of the 25$^{\text{th}}$ International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151.

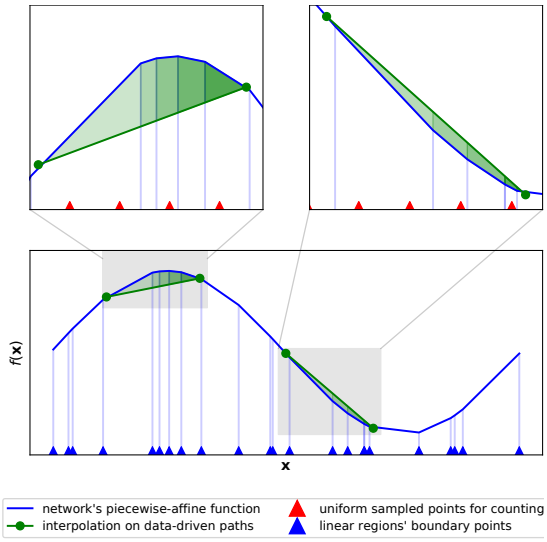


Figure 1: **General methodology:** we quantify the non-linearity of continuous piece-wise affine functions (blue line) for ReLU networks, which implicitly partition the input space into disjoint linear regions (bounded by blue triangles). Prior work used density of such regions as a proxy for nonlinearity. We note that 1) the slope and bias of each affine component, as well as the size of each region, affect nonlinearity, and 2) the same number of regions may correspond to different levels of nonlinearity (top box of the figure). Thus, we devise a novel measure based on absolute deviation from affine interpolation (green line) which better captures the non-linearity of learned functions (green shaded area). Furthermore, existing numerical methods measure density by sampling equidistant points (red triangles) which can be either prohibitively expensive or miss some regions depending on the granularity of sampling. Here, we propose an adaptive numerical algorithm for accurately counting all regions. Details of the proposed counting algorithm, the novel measure, and corresponding experiments are in sections 2.3, 2.4, and 3 respectively.

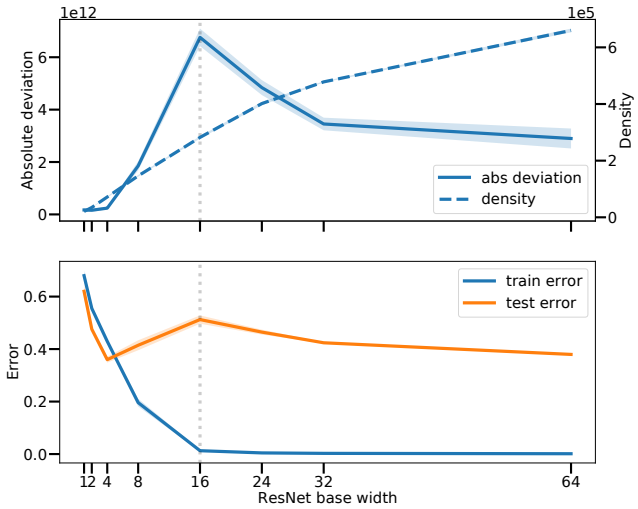


Figure 2: **Absolute deviation closely follows the test error past the interpolation threshold in a model-wise double descent regime.** At the same time, linear region density grows monotonically with model size rendering it unsuitable as a proxy for the complexity of a trained ReLU network. Thus, while larger models have in principle higher expressivity, their effective nonlinearity *decreases* during the second descent. (Top) Average median density and absolute deviation (over 3 seeds) computed on the CIFAR-10 training set with 20% noisy labels, for ResNets18s of increasing base width. (Bottom) Train and test error (0/1 loss) as a function of ResNet18 base width. The networks are trained with no explicit regularization or data augmentation, using Adam with base learning rate $1e-4$.

Our contributions [1] are summarized as follows.

- We propose an adaptive numerical algorithm for discovering linear regions along directions in the input space of trained deep networks (section 2.3). To our knowledge, ours is the first study estimating accurate linear region density for architectures such as ResNet (He et al., 2015). We present our findings on the CIFAR-10 and CIFAR-100 datasets (Krizhevsky et al., 2009).
- We contrast linear region density to a more principled way of estimating nonlinearity of piece-wise affine functions expressed by ReLU networks (section 2.4), accounting for the size of each linear region, as well as nonlinearity expressed by the respective affine components and their bias terms.
- Building on previous work (Novak et al., 2018), we study directions in the input space that meaningfully capture nonlinearity, but along which linear region density is unsuited for discerning generalizing networks from memorizing ones (section 3).
- We establish an empirical connection among test error, overparameterization, and nonlinearity, for which large models that harmlessly interpolate noise show lower nonlinearity as well as test error than models that harmfully overfit noise (section 3.6).

[1] Source code available at `https://github.com/magamba/linear-regions`

2 METHODOLOGY

We fix notation and describe existing approaches for estimating density of linear regions respectively in sections 2.1 and 2.2. In section 2.3, we present our linear region discovery algorithm. Finally, we introduce a simple measure of nonlinearity of continuous piecewise affine functions in section 2.4.

2.1 Notation

We consider ReLU networks as functions $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^K$, obtained by composing L affine layers, each with parameter matrix $\mathbf{W}^\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ and bias vector $\mathbf{b}^\ell \in \mathbb{R}^{d_\ell}$, for $\ell = 1, \dots, L$, with the continuous piece-wise affine activation function $\varphi(x) = \max(0, x)$, where $d_0 = d, d_L = K$, and φ is applied element-wise. The resulting function, $\mathbf{f}(\mathbf{x}) = \mathbf{W}^L \mathbf{x}^{L-1} + \mathbf{b}^L = \mathbf{W}^L \varphi(\dots \varphi(\mathbf{W}^1 \mathbf{x} + \mathbf{b}^1)) + \mathbf{b}^L$, is itself continuous piece-wise affine, and implicitly induces a partition $\mathcal{P}$ of the input space $\mathbb{R}^d$ into disjoint convex cells $A_\epsilon \in \mathcal{P}$ – generally referred to as linear regions – on which the network computes a single affine function $\mathbf{f}(\mathbf{x}) = \mathbf{W}_\epsilon \mathbf{x} + \mathbf{b}_\epsilon$, $\forall \mathbf{x} \in A_\epsilon$, for $\mathbf{W}_\epsilon \in \mathbb{R}^{K \times d}, \mathbf{b}_\epsilon \in \mathbb{R}^K$ (Balestriero et al., 2018; Montufar et al., 2014).

For an input $\mathbf{x} \in A_\epsilon$, the parameters of the corresponding affine component $\mathbf{W}_\epsilon \mathbf{x} + \mathbf{b}_\epsilon$, can be obtained by computing the product of matrices $\mathbf{W}_\epsilon = \prod_{\ell=1}^{L} \mathbf{W}_\epsilon^\ell$, as follows. For $\ell = 1, \dots, L$, each matrix $\mathbf{W}_\epsilon^\ell = \mathbf{W}^\ell \odot \mathbf{M}_\mathbf{x}^\ell$ is obtained by multiplying element-wise the weight matrix $\mathbf{W}^\ell$ of layer ℓ with the binary mask $\mathbf{M}_\mathbf{x}^\ell = \mathbf{M}^\ell(\mathbf{x}^{\ell-1}) = (M_{ij}) \in \{0,1\}^{d_\ell \times d_{\ell-1}}$, representing the activation pattern of ReLU at layer ℓ when $\mathbf{x}$ is input to the network, *i.e.* $M_{i,:} = 1$ if $(\mathbf{W}^\ell \mathbf{x}^{\ell-1} + \mathbf{b}^\ell)_i > 0$, and $M_{i,:} = 0$ otherwise. A similar procedure can be used for the bias parameter $\mathbf{b}_\epsilon = \sum_\epsilon \mathbf{b}_\epsilon^\ell$.

Then, using the indicator function, $\mathbf{f}$ can be decomposed as the sum over all linear regions of its affine components (Rahaman et al., 2019), yielding

$$\mathbf{f}(\mathbf{x}) = \sum_\epsilon \mathbb{1}_{\mathbf{A}_\epsilon}(\mathbf{x})(\mathbf{W}_\epsilon \mathbf{x} + \mathbf{b}_\epsilon) \quad \forall \mathbf{x} \in \mathbb{R}^d \qquad (1)$$

and making the locally affine behaviour of $\mathbf{f}$ explicit.

Next, we summarize prior work for estimating linear region density and highlight their limitations.

2.2 Empirical Estimates of Linear Region Density

Computational Complexity. The density of linear regions, defined as the size of the partition $\mathcal{P}$, has been used as a way to quantify nonlinearity of ReLU networks. In practice, exact calculation of $\mathcal{P}$ involves traversing the network sequentially, neuron by neuron, and computing how the hyperplanes defined by each neuron intersect with those at earlier layers (Zhang & Wu, 2020; Hanin & Rolnick, 2019a; Novak et al., 2018). Hence, estimating global density of overparameterized networks is prohibitively expensive in practice for high-dimensional input domains. Furthermore, the computational cost increases for convolutional layers whose weight tensors need unfolding into sparse weight matrices (Zhang & Wu, 2020).

Analytical Methods. To mitigate such computational challenges, prior work have resorted to calculating density of regions on 1-D and 2-D compact subsets $\mathcal{B} \subset \mathbb{R}^d$ of the input space, thus estimating the size of the restriction of $\mathcal{P}$ to $\mathcal{B}$. Exact analytical methods have so far been bound to small networks (Zhang & Wu, 2020; Hanin & Rolnick, 2019b,a) and simple datasets Hanin & Rolnick (2019b,b), making numerical approaches appealing.

Numerical Methods. Numerical methods have focused on bounded 1-dimensional trajectories in the input space, obtained by sampling equally-spaced points $\{\mathbf{x}_0, \dots, \mathbf{x}_n\}$, with a fixed step size $\lambda = \|\mathbf{x}_{i+1} - \mathbf{x}_i\|$, and exploiting the binary activation pattern $\left[\operatorname{vec}(\mathbf{M}_\mathbf{x}^1), \dots, \operatorname{vec}(\mathbf{M}_\mathbf{x}^L)\right]$ of each sample $\mathbf{x}$ as a signature for each region A_{ϵ_i}, thus estimating density by the number of unique such patterns (Novak et al., 2018; Raghu et al., 2017).

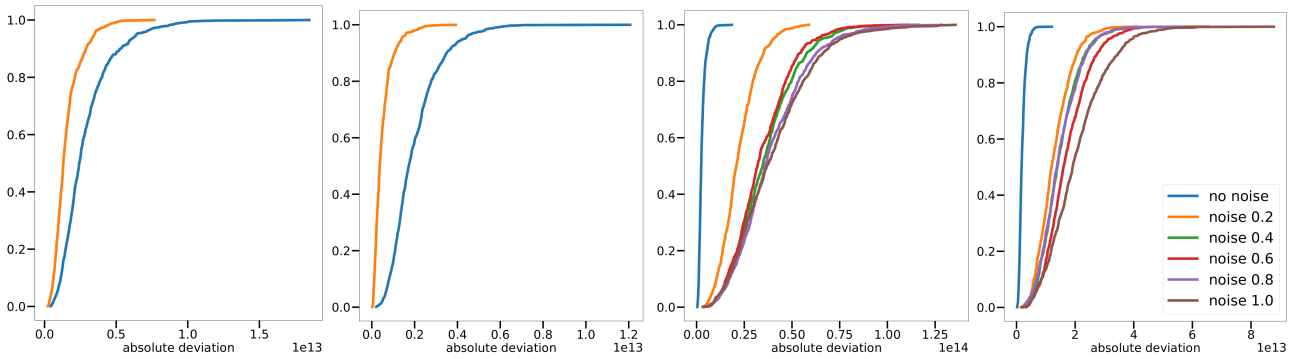
Shortcomings. Both analytical and numerical strategies attempt to capture nonlinearity of piece-wise affine functions by enumerating regions. On the one hand, the sequential nature of analytic approaches makes them unable to scale to realistic networks. On the other hand, numerical methods based on uniform sampling can only provide estimates on the number of regions by comparing activation patterns, without capturing the geometry of the underlying partition $\mathcal{P}$. Importantly, numerical approaches might potentially miss small regions when the step size λ is too large, or become prohibitively expensive for fine-grained sampling. Crucially, all existing strategies based on linear region density intrinsically assume all regions to equally contribute nonlinearity, independently of other properties of the corresponding affine function, as highlighted in Figure 1.

In the next section, we remove the dependency of numerical counting on a uniform step size λ, and instead provide a linear region discovery method that captures the size of linear regions along a given direction in the input space. Finally, in section 2.4, we introduce a simple measure that ties linear regions to nonlinearity of the corresponding piece-wise affine function.

2.3 Adaptive Linear Region Discovery

In this section, we present an adaptive numerical algorithm for discovering linear regions along a direction $\mathbf{d}$ in the input space, starting from an input point $\mathbf{x}_0$.

We begin by recalling that, for each affine layer ℓ with parameters $\mathbf{W}^\ell, \mathbf{b}^\ell$, each neuron j induces a hyperplane $\mathcal{H}_j^\ell : {\mathbf{w}_j^\ell}^T \mathbf{x} + b_j^\ell = 0$ in the preactivation space $\mathbb{R}^{d_{\ell-1}}$ of the layer, with $\mathbf{w}_j^\ell := \mathbf{W}_{j,:}^\ell$ and $b_j^\ell := \mathbf{b}_j^\ell$. In principle, given an input $\mathbf{x}_0$, belonging to linear



(a) (b) (c) (d)

Figure 3: Empirical cumulative distribution functions for absolute deviation along data-driven paths sampled from the CIFAR-10 training set, for several training settings. From the left, a) VGG8, and b) ResNet18, each trained with and without data augmentation. c) VGG8, and d) ResNet18 each trained on CIFAR-10 with increasingly noisy training labels. Absolute deviation captures increased smoothness for networks trained with data augmentation, as well as nonlinearity expressed by networks fitting increasingly harder tasks, thus constituting a principled baseline for studying linear region density in practical settings.

region A_ϵ, the boundaries of A_ϵ can be analytically determined as summarized in section 2.2. Here, we instead propose a faster numerical algorithm finding the closest linear region boundary of A_ϵ to $\mathbf{x}_0$, along $\mathbf{d}$.

Let $\mathbf{x}^{\ell-1} = \varphi(\mathbf{W}^{\ell-1}\mathbf{x}^{\ell-2} + \mathbf{b}^{\ell-1})$ denote the output of layer $\ell - 1$. Then, for each hyperplane $\mathcal{H}_j^\ell$ induced by layer ℓ, we solve the linear problem specified by Equation 3, to determine the closest boundary to $\mathbf{x}^{\ell-1}$ at layer ℓ in the direction $\mathbf{d}$. Assuming that $\mathbf{x}^{\ell-1}$ lies on the negative side of $\mathcal{H}_j^\ell$ [2], the smallest displacement λ_j^ℓ to cross $\mathcal{H}_j^\ell$ along $\mathbf{d}$ is computed by solving

$$\lambda_j^\ell = \min_{\substack{\lambda\in\mathbb{R} \\ |\lambda|>\tau}} \mathbf{w}_j^{\ell T}(\mathbf{x}^{\ell-1} + \lambda\mathbf{d}^\ell) + b_j^\ell > 0 \tag{2}$$

$$= \min_{\substack{\lambda\in\mathbb{R} \\ |\lambda|>\tau}} \lambda > \frac{-\mathbf{w}_j^{\ell T}\mathbf{x}^{\ell-1} - b_j^\ell}{\mathbf{w}_j^{\ell T}\mathbf{d}^\ell} \tag{3}$$

where $\mathbf{d}^\ell$ represents the direction $\mathbf{d}$ in the preactivation space $\mathbb{R}^{d_{\ell-1}}$ of layer ℓ, and is obtained as the product $\prod_{p=1}^{\ell-1} \mathbf{W}_\epsilon^p \cdot \mathbf{d}$, where the matrices $\mathbf{W}_\epsilon^p$ depend on the activation pattern of linear region A_ϵ. To control numerical stability and guarantee that the nearest linear region boundary is always crossed, the solution is computed for $|\lambda| > \tau$, with $\tau \ll 1$, acting as a sensitivity parameter, ensuring that a minimal step along $\mathbf{d}$ is taken at each iteration of the algorithm. [3]. Moreover, the linear problem has no solution if $\mathbf{d} = \mathbf{0}$, or $\mathbf{d}$ lies on $\mathcal{H}_j^\ell$, in which case the inequality is discarded.

Given a starting point $\mathbf{x}_0 \in \mathbb{R}^d$, and a direction vector $\mathbf{d}$, the smallest displacement $\lambda = \min_{1\le j\le d_{\ell-1}, 1\le\ell\le L} \lambda_j^\ell$ is computed by a single forward pass of $\mathbf{x}_0$ and $\mathbf{d}$, solving the linear problem 3 iteratively for each layer.

Pseudocode for our algorithm is included in section C.

Density of Compact 1-D Domains Iterating the procedure, starting from $\mathbf{x}_0$, it is possible to find all linear regions along a direction $\mathbf{d}$, which we parameterize as a line path $\boldsymbol{\pi} : \mathcal{I} = [0, 1] \to \mathbb{R}^d$, with $\boldsymbol{\pi}(t) = \mathbf{x}_0 + t\mathbf{d}$, for $0 \le t \le 1$. Crucially, by convexity of linear regions, $\mathcal{P}$ induces a partition $\mathcal{P}_\mathcal{I} = \{t_\epsilon \in \mathcal{I} : 0 = t_0 < \ldots < t_D = 1\}$ of $\mathcal{I}$, dividing $\mathcal{I}$ into intervals $\mathcal{J}_\epsilon = [t_\epsilon, t_{\epsilon+1}]$, with $|\mathcal{J}_\epsilon| = \lambda_\epsilon$ corresponding to the length of region A_ϵ along $\mathbf{d}$.

Importantly, this method allows us to compute the entry and exit point of $\boldsymbol{\pi}$ into each linear region along $\mathbf{d}$, and returns exactly all linear regions of size greater than τ. In the following, we use such quantities to introduce a simple measure of nonlinearity of a network.

2.4 Quantifying Nonlinearity

In principle, as illustrated in Figure 1, nonlinearity of continuous piece-wise affine functions is controlled by the slope and bias of each component, as well as the volume of the corresponding linear region. Crucially,

[2] If instead $\mathbf{x}^{\ell-1}$ lies on the positive side of $\mathcal{H}_j^\ell$, the inequality is flipped, as well as the sign of λ_j^ℓ.

[3] If the size of a linear region along $\mathbf{d}$ is lower than τ, then τ effectively acts as a step size. In practice, choosing $\tau = 1\text{e}-6$ with double precision computation resulted in 2% of linear regions being found with size lower or equal than τ, negligibly affecting our comparisons.

for overparameterized networks, several neighbouring regions could encode approximately the same affine function, making density unsuited for capturing nonlinearity, interpreted as effective complexity.

To investigate the effectiveness of density at capturing nonlinearity of ReLU networks, we hereby introduce a simple notion of nonlinearity along a direction – called absolute deviation – measuring the deviation of a ReLU network $\mathbf{f}$ from a simple function $\mathbf{a}$ interpolating the affine components respectively applied by $\mathbf{f}$ at the endpoints of a line path $\boldsymbol{\pi}$ in the input space.

For two distinct inputs $\mathbf{x}_0, \mathbf{x}_1 \in \mathbb{R}^d$, respectively falling into linear regions A_0 and A_1, with corresponding affine transformations $\mathbf{f}_0(\mathbf{x}) = \mathbf{W}_0\mathbf{x} + \mathbf{b}_0$, and $\mathbf{f}_1(\mathbf{x}) = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$, we define the affine function $\mathbf{a}(\mathbf{x})$ obtained by linearly interpolating the functions $\mathbf{f}_0$ and $\mathbf{f}_1$, so that, along a line path $\boldsymbol{\pi}$ parameterizing $\mathbf{d} = \mathbf{x}_1 - \mathbf{x}_0$, it holds $\mathbf{a}(\mathbf{x}) = \mathbf{f}(\mathbf{x}_0) + t(\mathbf{f}(\mathbf{x}_1) - \mathbf{f}(\mathbf{x}_0))$. That is, we only specify $\mathbf{a}$ so that it interpolates $\mathbf{f}(\mathbf{x}_0)$ and $\mathbf{f}(\mathbf{x}_1)$ along $\mathbf{d}$. Importantly, the difference $\mathbf{f}(\mathbf{x}_1) - \mathbf{f}(\mathbf{x}_0)$ is in general a nonlinear affine function, *i.e.* with non-zero bias term.

Then, for each output dimension f^k of the network, for $k = 1, \dots, K$, we can measure the nonlinearity of f^k along a line path $\boldsymbol{\pi}$ by computing

$$
\begin{aligned}
&\int_{\boldsymbol{\pi}} |f^k(\mathbf{x}) - a^k(\mathbf{x})| d\mathbf{x} \\
&= \sum_{\epsilon=0}^{D-1} \int_{t_\epsilon}^{t_{\epsilon+1}} |f^k(\boldsymbol{\pi}(t)) - a^k(\boldsymbol{\pi}(t))| \cdot \|\dot{\boldsymbol{\pi}}(t)\| dt \\
&= \sum_{\epsilon=0}^{D-1} \int_{t_\epsilon}^{t_{\epsilon+1}} |f^k(\mathbf{x}_0 + t\mathbf{d}) - a^k(\mathbf{x}_0 + t\mathbf{d})| \cdot \|\mathbf{d}\| dt \\
&= \|\mathbf{d}\| \cdot \sum_{\epsilon=0}^{D-1} \int_{t_\epsilon}^{t_{\epsilon+1}} \big(|f_\epsilon^k(\mathbf{x}_0) - f_0^k(\mathbf{x}_0) + \\
&\qquad t\big(f_\epsilon^k(\mathbf{x}_1) - f_\epsilon^k(\mathbf{x}_0) - f_1^k(\mathbf{x}_1) + f_0^k(\mathbf{x}_0)\big)| \big) dt
\end{aligned}
\tag{4}
$$

where $f_\epsilon^k(\mathbf{x}_i) := (\mathbf{W}_\epsilon)_{k,:}\, \mathbf{x}_i + (\mathbf{b}_\epsilon)_k$, *i.e.* the k-th output logit of the affine component ϵ of $\mathbf{f}$, evaluated at $\mathbf{x}_i$. A full derivation is presented in section D.

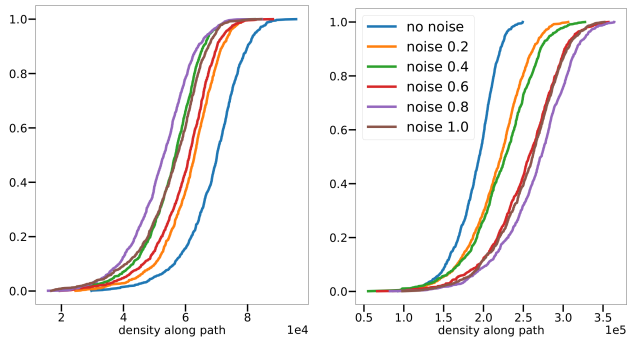
Referring again to Figure 1, we observe that, 1) since $t_{\epsilon+1} - t_\epsilon = \lambda_\epsilon$, absolute deviation (shaded green area) precisely follows the piece-wise affine structure of $\mathbf{f}$, taking into account the length of linear regions A_ϵ along $\mathbf{d}$, as well as the slope of $\mathbf{f}$; 2) absolute deviation takes into account all linear regions along a trajectory $\boldsymbol{\pi}$, providing an accurate benchmark of expressivity against linear region density; 3) in contrast to total variation measures (Novak et al., 2018), the integrand depends on the bias of $\mathbf{f}$, providing a more precise approach than solely estimating any $\mathbf{W}_\epsilon$ through gradient information $\nabla_{\mathbf{x}}\mathbf{f}$, as nonlinearity arising from bias terms is otherwise lost; 4) the integrand smoothly interpolates between the affine function f_0^k and f_1^k such that, if $\mathbf{f}$ computes affine functions that are approximately similar to $\mathbf{f}(\mathbf{x}_0)$ or $\mathbf{f}(\mathbf{x}_1)$ along $\boldsymbol{\pi}$, then the resulting absolute deviation will be low; 5) if $\mathbf{x}_0, \mathbf{x}_1$ are training or validation points, then $\boldsymbol{\pi}$ anchors $\mathbf{a}$ to the support of the data-generating distribution at $\mathbf{x}_0$ and $\mathbf{x}_1$, and allows for computing nonlinearity in proximity of the where the data lies in the input space.

Lastly, we note that absolute deviation is a one-dimensional estimate of nonlinearity, meant to contrast density of linear regions, rather than to stand as a novel generalization measure. In particular, its effectiveness at capturing nonlinearity expressed by learning is bound to evaluating the measure along meaningful directions $\mathbf{d}$ in the input space. In section 3, we exploit data-driven trajectories introduced in (Novak et al., 2018), which were shown to capture sensitivity of $\mathbf{f}$ to input perturbations.

3 EXPERIMENTS

In this section, we empirically investigate practical training settings in which density of linear regions is an unreliable estimator of nonlinearity. We apply our linear region discovery algorithm to $N = 1024$ closed paths $\boldsymbol{\pi}_n$ in the input space of trained networks, constructed by connecting a training sample $\mathbf{x}_n^0$, with augmented versions obtained by deterministically translating $\mathbf{x}_n^0$ along a circular trajectory (Novak et al., 2018), which we define of radius 4 to reflect common data-augmentation strategies. Details on the construction are presented in section E. Each closed path is defined using $A = 8$ augmented samples, each connected to the next one along the circular trajectory, by straight lines $\mathbf{d}_n^a := \mathbf{x}_n^{(a+1)\%A} - \mathbf{x}_n^a$, for $a = 0, \dots A$. Each point $\mathbf{x}_n^a$ anchors the path to the support of the data distribution, ensuring that density and deviation along each line $\mathbf{d}_n^a$ are evaluated in proximity of the data manifold in pixel space. Starting from a base point $\mathbf{x}_n^0$ for each path $\boldsymbol{\pi}_n$, we compute the entry and exit point of $\boldsymbol{\pi}_n$ for each linear region it crosses, and compute linear region density as well as absolute deviation, for several trained networks.

We perform our experiments on the CIFAR-10 and CIFAR-100 datasets (Krizhevsky et al., 2009), using a VGG-like network (Simonyan & Zisserman, 2015) with 8 layers, and a ResNet18 (He et al., 2015) with base width 16. All measures are computed on the training split of the dataset considered. All networks fitting noisy labels are trained until 100% training accuracy is reached. Our experimental setup and network architectures are detailed in sections A and B.



(a) (b)

Figure 4: ECDF of density along data-driven paths on CIFAR-10 samples, for increasingly noisy labels. Importantly, it can be observed that a fixed density value can realize functions of different nonlinearity. a) VGG8. b) ResNet18.

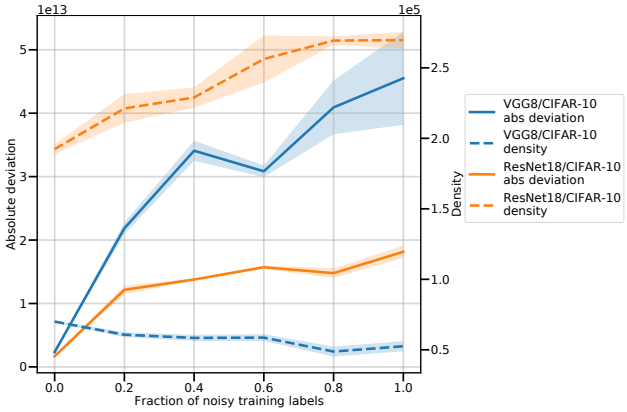


Figure 5: Median density and absolute deviation for increasingly noisy learning tasks, averaged over three independent runs. While absolute deviation almost always increases for harder problems, density decreases for ResNet18 and increases for VGG8, thus providing a noisy predictor of nonlinearity.

Finally, we recall that, for a network with K-dimensional output, absolute deviation produces K scalar scores – one per logit. Throughout our experiments, we interpret such scores together as a K-dimensional vector, of which we take the ℓ_2 norm to produce a single scalar measure per path, which can be contrasted to linear region density along the same path. We keep the set of paths fixed throughout our experiments, in order to study how our measures vary when the training setup is changed.

3.1 Absolute Deviation Captures Complexity of Trained Networks

We begin by evaluating whether absolute deviation is a suitable measure of nonlinearity, by studying networks trained on learning tasks with increasingly noisy training labels, pushing each network towards learning increasingly nonlinear decision functions. In Figure 3, we compare the functions parameterized by VGG8 and ResNet18 on CIFAR-10 for each training setting, by studying the Empirical Cumulative Distribution Function (ECDF) of absolute deviation evaluated on the N paths. Furthermore, we evaluate whether absolute deviation is able of capturing smoothness of networks in the input variable $\mathbf{x}$, as promoted by data augmentation. For both VGG and ResNet, in the noisy labels setting, we observe that absolute deviation is concentrated towards relatively low values for networks trained on clean labels, while higher deviation is realized by networks fitting harder tasks, with the 100% noisy labels setting realizing highest deviation. Furthermore absolute deviation clearly separates piecewise affine functions regularized with data augmentation from those trained in vanilla settings with no augmentation. Therefore, we conclude that our novel metric is a suitable candidate for assessing the ability of linear region density to capture nonlinearity for networks trained in practice.

For comparison, Figure 4 shows the ECDF for density along the same paths and training settings. Absolute deviation better separates noisy labels training settings for clean labels, by concentrating probability mass towards low values for less-noisy datasets. Furthermore, while VGG8 realizes higher density for clean labels than for any noisy setting, ResNet18 expresses more regions for noisy labels, showing that density is an unreliable predictor of expressivity.

3.2 Absolute Deviation Better Ranks Nonlinearity

Next, we investigate how absolute deviation and density fare at ranking learning tasks of increasing complexity, represented by increasing fraction of noisy training labels on CIFAR-10. Figure 5 shows the average median density and median absolute deviation, as a function of the fraction of noisy labels, with each median averaged over three independent training runs. Absolute deviation correctly separates networks trained on clean labels from all noisy labels setting, and correctly ranks networks trained on 80% and 100% noisy labels as the most nonlinear, while at the same time fails to separate between 40% and 60% noise for VGG, and 60% and 80% for ResNet. Strikingly, density increases with task complexity for ResNet18, but decreases for VGG8, showing that, along the data-driven directions considered, density is a fragile esti-

Table 1: Spearman rank correlation between density and absolute deviation along data-driven paths. Consistently, density poorly correlates with absolute deviation, sometimes anticorrelating.

	CIFAR-10		CIFAR-100
	VGG8	ResNet18	VGG8
vanilla	-0.15 ± 0.00	0.05 ± 0.03	0.00 ± 0.02
augment.	-0.08 ± 0.02	0.17 ± 0.06	0.11 ± 0.08
noise 0.2	0.04 ± 0.02	0.21 ± 0.02	
noise 0.4	0.11 ± 0.02	0.21 ± 0.04	
noise 0.6	0.18 ± 0.07	0.25 ± 0.03	
noise 0.8	0.26 ± 0.04	0.22 ± 0.02	
noise 1.0	0.14 ± 0.04	0.27 ± 0.04	

mator of nonlinearity, and that a more precise measure of variation of the piece-wise affine function itself offers a more robust measure.

3.3 Density Poorly Correlates with Absolute Deviation

Recalling that absolute deviation is expressed as a sum of non-negative terms over linear regions (Equation 4), we investigate whether it can simply be explained by density. For ResNet18 and VGG8 trained on CIFAR-10 and CIFAR-100, we compute the Spearman rank correlation coefficient between density and deviation, across N fixed data-driven paths. Table 1 reports the average correlation coefficients with one standard deviation, computed over three independent training runs for each network and training setting. Consistently, density correlates poorly with absolute deviation, presenting negative correlations for VGG trained without data augmentation on both datasets.

3.4 Density May Fail to Detect Increased Nonlinearity

We now move from a distribution-level study to an instance-based analysis, investigating how density and absolute deviation change at the level of individual data-driven paths, when comparing pairs of training settings. Specifically, for a set of N paths $\{\boldsymbol{\pi}_n\}_{n=1}^N$, and a pair of training settings (T_1, T_2), we collect corresponding measures of deviation $\{\mathrm{dev}_n^i\}_{n=1}^N$ and density $\{\mathrm{den}_n^i\}_{n=1}^N$, for $i = 1, 2$, and compute paired differences $\mathrm{dev}_n^2 - \mathrm{dev}_n^1$, $\mathrm{den}_n^2 - \mathrm{den}_n^1$, for each $n = 1, \ldots, N$. The, for pairs of training settings (T_1, T_2) sorted so that T_2 entails learning a function with higher nonlinearity than T_1, the ability of density and absolute deviation of capturing nonlinearity at the level of individual paths can be measured by the fraction of positive paired differences observed, *i.e.* $|\{\mathrm{dev}_n^2 - \mathrm{dev}_n^1 > 0 \text{ for } n = 1, \ldots, N\}|$. Table 2 collects our experimental findings, averaged over three independent training runs for each setting.

First, we observe that deviation is able to reliably and consistently detect increased nonlinearity at the level of individual paths, capturing additional nonlinearity expressed by networks when going from no noisy labels to any amount of noisy labels, as well as when disabling data augmentation. In contrast, density fails to do the same, in the same settings, on both CIFAR-10 and CIFAR-100. Second, when comparing each trained network to the same network at initialization, across all training settings, both deviation and density are able to capture nonlinearity arising from learning, showing that the density of linear regions is on average higher for trained networks. However, even in this setting, density fails to capture nonlinearity for up to 20% paths, proving to be a fragile measure in practice.

3.5 Density Fails to Distinguish Piece-Wise Affine Functions

Next, we investigate whether density and absolute deviation, measured on training data, can predict the network's test error. In Figure 6, we begin by collecting the test error for all networks, datasets, and training configurations considered in this study. In line with what reported by Novak et al. (2018), extending their result to convolutional and residual networks, we observe that measures of variation, like absolute deviation, can better predict test error, here measured using the 0/1 loss. Importantly, we observe how any fixed value of density can result in networks of vastly different test performance, showing how merely counting the number of affine components is unable to measure effective complexity of continuous piece-wise affine functions parameterized by ReLU networks.

3.6 Overparameterization Reduces Effective Nonlinearity

To further explore the connection between nonlinearity and test error, we note that on the one hand, higher expressivity is in principle afforded by increased model size (Telgarsky, 2016; Montufar et al., 2014), while on the other hand overparameterization can promote regularization, observed in the form of reduced test error, in model-wise deep double descent regimes (Nakkiran et al., 2019; Belkin et al., 2019).

We reproduce the experimental setting of Nakkiran et al. (2019) (*cfr.* Figure 4b), and study ResNet18s of increasing base widths w up to $w = 64$ (standard ResNet18), on CIFAR-10 with 20% noisy training labels. All networks are trained for 400 epochs, using Adam with base learning rate $1\mathrm{e}-4$, without any explicit regularization or any data augmentation. We re-

Table 2: Paired differences for several training settings, averaged over three independent training runs for each training setting.

	CIFAR-10				CIFAR-100	
	VGG8		ResNet18		VGG8	
	Abs deviation	Density	Abs deviation	Density	Abs deviation	Density
vanilla vs augment.	0.98 ± 0.01	0.66 ± 0.40	1.00 ± 0.00	0.03 ± 0.04	1.00 ± 0.00	0.38 ± 0.24
noise 0.2 vs no noise	1.00 ± 0.00	0.00 ± 0.00	1.00 ± 0.00	0.95 ± 0.02		
noise 0.4 vs no noise	1.00 ± 0.00	0.00 ± 0.00	1.00 ± 0.00	0.78 ± 0.02		
noise 0.6 vs no noise	1.00 ± 0.00	0.00 ± 0.01	1.00 ± 0.00	0.86 ± 0.03		
noise 0.8 vs no noise	1.00 ± 0.00	0.00 ± 0.00	0.99 ± 0.00	0.91 ± 0.02		
noise 1.0 vs no noise	1.00 ± 0.00	0.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00		
vanilla trained vs init.	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.79 ± 0.05	1.00 ± 0.00	1.00 ± 0.00
augment. trained vs init.	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.99 ± 0.01	1.00 ± 0.00	1.00 ± 0.00
noise 0.2 trained vs init.	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.82 ± 0.04		
noise 0.4 trained vs init.	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.96 ± 0.01		
noise 0.6 trained vs init.	1.00 ± 0.00	0.99 ± 0.01	1.00 ± 0.00	0.99 ± 0.01		
noise 0.8 trained vs init.	1.00 ± 0.00	0.96 ± 0.03	1.00 ± 0.00	1.00 ± 0.00		
noise 1.0 trained vs init.	1.00 ± 0.00	0.96 ± 0.01	1.00 ± 0.00	0.94 ± 0.01		

port absolute deviation and density in Figure 2 (top) as a function of model size, as well as the train and test error (bottom), measured with the 0/1 loss.

In line with Nakkiran et al. (2019), ResNets18s of increasing width first realize decreasing train and test error ($w \leq 4$), then overfit the noisy training set incurring in increased test error ($4 < w \leq 16$), up to an *interpolation threshold* ($w = 16$), at which zero train error is achieved, with highest test error. Then, a second descent of the test error occurs ($w > 16$), while the networks still perfectly interpolate the training data. Intriguingly, absolute deviation is low for underfitting networks ($w \leq 4$, and train error $> 30\%$), which fail to interpolate noise; then the measure increases, peaking at the interpolation threshold, to finally decrease again following the second descent of the test error.

In stark contrast, density monotonically increases with model size. Taken together with absolute deviation, this finding shows that, while larger networks can afford greatly many linear regions, overparameterization past the interpolating threshold induces regularization in the form of reduced nonlinearity, *i.e.* increased local similarity of linear regions. To our knowledge, this is the first experiment explicitly connecting reduced test error in the model-wise double descent regime with reduced nonlinearity of the learned function.

4 RELATED WORK

Density of Linear Regions Linear regions were introduced by Pascanu et al. (2013) and Montufar et al. (2014), to make sense of hierarchical representations in ReLU networks, illustrating how they nonlinearly partition the input space into disjoint cells, and providing lower bounds on density realized by a given architecture. Several works followed, proving refined bounds on density (Xiong et al., 2020; Hanin & Rolnick, 2019a; Arora et al., 2018; Serra et al., 2018; Raghu et al., 2017). Hanin & Rolnick (2019b) formally study neural networks at initialization, providing average-case bounds over the weight space of MLPs on the local density of linear regions, computed on bounded volumes. Beyond initialization, they hypothesize that the implicit bias of SGD pushes learning towards functions that realize relatively low density – expressed by a polynomial bound in the number of neurons.

Novak et al. (2018) estimate sensitivity of dense ReLU networks to input perturbations, as well as density of linear regions, as two independent quantities, along data-driven directions in the input space of trained networks. They observe that while their measure of sensitivity correlates with generalization, density does so only weakly. Similarly, in this work we adopt the same strategy for generating data-dependent directions in the input space, but use them to carry out a systematic investigation of the relationship between density of regions and the nonlinearity of the piecewise affine functions. Importantly, while Novak et al. (2018) estimate density by sampling uniformly in the input space, our work accounts for the anisotropic geometry of linear regions, by providing an adaptive algorithm for region discovery, thus improving on the tradeoff between analytical precision and scalability.

Density-agnostic Works on Bounded Variation for Generalizing Networks Recent works deals with measures of variation implicitly regularized by optimization, by explicitly accounting for linear regions, but not for their density. Rahaman et al. (2019) identify an implicit bias of SGD, by comput-

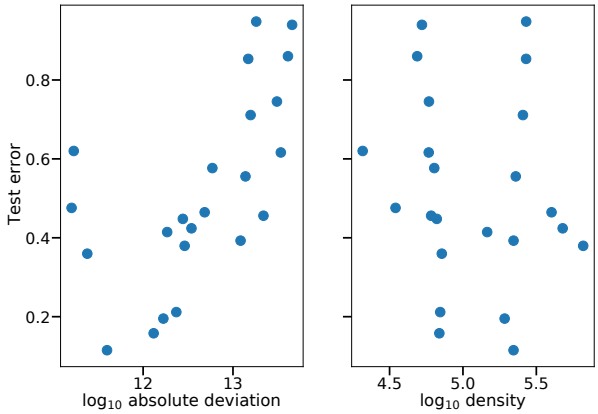


Figure 6: Mean test error against mean absolute deviation (left) and mean density (right), computed on the training set, for all our experimental settings. Means are computed over three independent runs (x-axis shown in log-scale).

ing the Fourier transform of ReLU networks, and find directions tied with slower spectral decay of ReLU networks. While their work directly exploits the piecewise linearity of ReLU networks, as well as the notion of linear region, their effort is focused on establishing the emergence of the spectral bias.

LeJeune et al. (2019) study the complexity of ReLU networks, by introducing a *rugosity* measure based on the tangent Hessian of the function. Their work highlights how commonly used data augmentation implicitly regularizes rugosity. While their work is not directly concerned with the density of linear regions, they identify as meaningful future work the study of variation of piece-wise affine networks, and speculate whether such notion would be more appropriate to study the implicit bias of SGD, as opposed to directly enumerating regions. Our work follows in spirit their observation, and we are able to empirically identify directions in the input space in which uniform density of regions fails to correlate with variation of the underlying function, which itself correlates with learning.

Importantly, existing function variation measures have so far not been studied in the double descent regime, in relation to test error and increased overparameterization. Our work is the first to explicitly tie reduced test error with reduced nonlinearity of ReLU networks.

Empirical Properties of Linear Regions Zhang & Wu (2020) carry out an empirical study of linear regions for small networks trained in practice, relating statistics of linear regions with several neural network hyperparameters. Furthermore, they study how explicit regularization affects similarity of neighbouring linear regions using validation points, and observe that neighbouring regions equally contribute to the prediction of the network. Their study focuses on the effect of different explicit regularization techniques on the statistics of linear regions, while we relate function variation with realized density of linear regions.

Trimmel et al. (2021) provide an algorithm for compressing a trained network by using only non-empty linear regions containing training data, which is able to partly recover the performance of the uncompressed network. The authors argue for their extraction method to be better suited at capturing nonlinearity underlying piece-wise affine functions in place of density of linear regions, but do not investigate where density fails in practice. Our work is complementary to theirs in research question and methodology, as we systematically assess the fragility of linear region density as an estimator of nonlinearity for trained networks.

5 CONCLUSIONS

Our experimental investigation, contrasting density of regions to a principled measure of nonlinearity, shows that not all linear regions equally contribute to complexity underlying learning in ReLU networks. Crucially, following model-wise double descent, linear regions increase in local similarity, with corresponding reduction of both nonlinearity and harmful overfitting. Existing density estimates, that equally weigh different linear regions, provide a fragile measure in practice for modern overparameterized networks. While density still serves as an important tool to describe theoretical expressivity of ReLU network architectures, our work supports with empirical evidence the criticism of density as a complexity measure for trained networks. Importantly, our work highlights that estimates of locally linear behaviour for ReLU networks can correlate with complexity and learning. Thus, implicit bias of SGD should be sought in bounded variation of the learned function, rather than by bounding uniform density of regions. We leave to future work the question of further biasing region-counting in the input space, so that redundant regions are weighted less, potentially resulting in a more robust notion of density.

Acknowledgments

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, as well as partially funded by Swedish Research Council project 2017-04609. Large-scale experiments were enabled by resources provided by the Swedish National Infrastructure for Computing (SNIC) at Chalmers Centre for Computational Science and Engineering (C3SE) partially funded by the

Swedish Research Council through grant agreement no. 2018-05973, as well as by resources provided by the National Supercomputer Centre (NSC), funded by Linköping University.

References

Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. In *International Conference on Learning Representations*, 2018.

Randall Balestriero et al. A spline theory of deep learning. In *International Conference on Machine Learning*, pp. 374–383. PMLR, 2018.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116 (32):15849–15854, 2019.

Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. In *29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pp. 698–728, Columbia University, New York, New York, USA, 23–26 Jun 2016. PMLR.

Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. In *International Conference on Machine Learning*, pp. 2596–2604, 2019a.

Boris Hanin and David Rolnick. Deep relu networks have surprisingly few activation patterns. In *Advances in Neural Information Processing Systems*, pp. 359–368, 2019b.

Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pp. 6–20. ACM, 1986.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015.

Kenji Kawaguchi, Leslie Pack Kaelbling, and Yoshua Bengio. Generalization in deep learning. *arXiv preprint arXiv:1710.05468*, 2017.

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

Daniel LeJeune, Randall Balestriero, Hamid Javadi, and Richard G Baraniuk. Implicit rugosity regularization via data augmentation. *arXiv preprint arXiv:1905.11639*, 2019.

Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 2924–2932, 2014.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *International Conference on Learning Representations*, 2019.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. In *International Conference on Learning Representations Workshop Track*, 2015.

Roman Novak, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. In *International Conference on Learning Representations*, 2018.

Razvan Pascanu, Guido F Montufar, and Yoshua Bengio. On the number of inference regions of deep feed forward networks with piece-wise linear activations. 2013.

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In *International Conference on Machine Learning*, pp. 2847–2854, 2017.

Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 5301–5310. PMLR, 09–15 Jun 2019.

Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. In *International Conference on Machine Learning*, pp. 4558–4566, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

Matus Telgarsky. Benefits of depth in neural networks. In *29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pp. 1517–1539, Columbia University, New York, New York, USA, 23–26 Jun 2016. PMLR.

Martin Trimmel, Henning Petzka, and Cristian Sminchisescu. Tropex: An algorithm for extracting linear terms in deep neural networks. In *International Conference on Learning Representations*, 2021.

Huan Xiong, Lei Huang, Mengyang Yu, Li Liu, Fan Zhu, and Ling Shao. On the number of linear regions of convolutional neural networks. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 10514–10523. PMLR, 13–18 Jul 2020.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *International Conference on Learning Representations*, 2018.

Hongyi Zhang, Yann N. Dauphin, and Tengyu Ma. Residual learning without normalization via better initialization. In *International Conference on Learning Representations*, 2019.

Xiao Zhang and Dongrui Wu. Empirical studies on the properties of linear regions in deep neural networks. In *International Conference on Learning Representations*, 2020.

Supplementary Material: Are All Linear Regions Created Equal?

The following presents a detailed description of our methodology and experimental setting. Section A describes our experimental setup, with training hyperparameters and hardware used. Section B describes the network architectures used. Pseudocode for our linear region discovery algorithm is presented in section C, while section D provides a full derivation of absolute deviation. Then, section E details the transformations used to generate data-driven trajectories in the input space. Finally, section F presents additional experiments.

A Training Hyperparameters

With the exception of Figure 2, all networks are trained using SGD with momentum 0.9 and batch size 128, on the CIFAR-10 and CIFAR-100 datasets. Pixel values are normalized using per-channel mean and standard deviation, computed on the full training split of the corresponding dataset. For each dataset, training hyperparameters are selected on a fixed validation split of size 1000, randomly sampled from the training set. All networks are trained without any explicit regularization enabled (dropout, batch normalization, weight decay), save where specified. For all training settings we run 3 random seeds, controlling random weight initialization and the ordering of samples during training.

Vanilla and Augmentation Training Settings To ensure a fair computational budget across different network architectures, convergence criteria are set on CIFAR-10 and CIFAR-100 respectively so that all networks are trained until the training cross entropy loss falls below 0.19, and 0.25. For all networks, an initial learning rate of 0.1 is decayed by a factor of 0.2 every 150 epochs. When data augmentation is enabled, training images are randomly shifted by 4 pixels and randomly flipped horizontally. To avoid overfitting, all networks are trained with weight decay $5\mathrm{e}-4$.

Overfitting Noisy Labels On noisy labels, all networks are trained until 100% training accuracy is reached, following the experimental setup of Zhang et al. (2018). Namely, an initial learning rate of 0.1 is decayed with a multiplicative factor of 0.95 at every epoch. For this setting, no data augmentation is used.

Model-Wise Deep Double Descent Following the experimental setting of Nakkiran et al. (2019), we train ResNets18 of increasing base width $w \in \{1, 2, 4, 8, 16, 24, 32, 64\}$, according to the ResNetv1 architecture (He et al., 2015), consisting of residual blocks grouped in 4 stages, with respective width $[w, 2w, 4w, 8w]$. All networks are trained for 400 epochs with Adam with base learning rate $1\mathrm{e}-4$ on CIFAR-10 with 20% corrupted training labels. We limit the number of training epochs to 400 to avoid incurring in epoch-wise double descent, and focus on model-wise descent only. All networks are trained with no explicit regularization (especially without batch normalization), nor data augmentation. To ensure stable training, FixUp initialization is used (Zhang et al., 2019), without the affine rescaling transformations to avoid altering the ResNet architecture.

Hardware Infrastructure Our experiments on VGG8 are performed using 8 NVIDIA V100s with 32GB of memory, while for ResNet18, linear region counting is performed using 8 A30s, each with 40GB of memory.

B Network Architectures

Table 3 lists all network architectures used. VGG is initialized following the He initialization scheme (He et al., 2015), with bias parameters initialized to zero. ResNet is instead initialized using a modified version of FixUp (Zhang et al., 2019), where in place of initializing some layers to zero, each component is sampled i.i.d. from a Gaussian distribution of mean zero and standard deviation $1\mathrm{e}-6$. To preserve piece-wise linearity of the functions parameterized by the networks, average pooling was used in place of max pooling. We note that the

Table 3: Network architectures used in our experiments. Following the notation of Simonyan & Zisserman (2015), Conv3-64 denotes a convolutional layer of kernel size 3×3 learning 64 feature maps. Strides greater than 1 are denoted by $//s$, e.g. $//2$ for stride 2. For both architectures, each layer save for the last linear one is followed by a ReLU activation. For ResNet18, "downsample" convolutions denote residual connections with stride larger than 1. All trained layers learn bias parameters, save for the 1×1 convolutions in the ResNet downsample blocks.

	VGG8	ResNet18
Input size	$32 \times 32 \times 3$	$32 \times 32 \times 3$
	Conv3-6 Conv3-6 AveragePool (2,2)	Conv3-16
Input size	$16 \times 16 \times 6$	$32 \times 32 \times 16$
	Conv3-16 Conv3-16 AveragePool (2,2)	Conv3-16 Conv3-16 Conv3-16 Conv3-16
Input size	$8 \times 8 \times 16$	$32 \times 32 \times 16$
	Conv3-64 Conv3-64 AveragePool (2,2)	Conv3-32 $//2$ Conv3-32 Downsample: Conv1-32 $//2$ Conv3-32 Conv3-32
Input size	$4 \times 4 \times 64$	$16 \times 16 \times 32$
		Conv3-64 $//2$ Conv3-64 Downsample: Conv1-64 $//2$ Conv3-64 Conv3-64
Input size		$8 \times 8 \times 64$
		Conv3-128 $//2$ Conv3-128 Downsample: Conv1-128 $//2$ Conv3-128 Conv3-128
Input size	$4 \times 4 \times 64$	$4 \times 4 \times 128$
	Global Average Pooling (4, 4)	
Input size	128	128
	fc-1024, fc-120	fc-128
Depth	8	18
Number of parameters	174116 (CIFAR-10) 185006 (CIFAR-100)	700042 (CIFAR-10)

Algorithm 1 Find minimum displacement λ to cross into the next linear region.

function FIND-LAMBDA($\mathbf{f}, \mathbf{x}, \mathbf{d}$)
  $\lambda \leftarrow \infty$
  for $\ell = 1, \ldots, L$ **do** ▷ Iterate sequentially through all layers.
    lambdas $\leftarrow \emptyset$ ▷ Set of candidate lambdas for layer ℓ.
    lambdas $\leftarrow \frac{-\mathbf{b}^\ell - \mathbf{W}^\ell \mathbf{x}}{\mathbf{W}^\ell \mathbf{d}}$ ▷ Element-wise division, producing d_ℓ candidate lambdas.
    lambdas[lambdas $< \tau$] $\leftarrow \infty$ ▷ Filter out numerically unstable lambdas.
    if min lambdas $< \lambda$ **then**
      $\lambda \leftarrow$ min lambdas
    end if
    $\mathbf{x} \leftarrow \mathbf{W}_\epsilon^\ell \mathbf{x} + \mathbf{b}_\epsilon^\ell$ ▷ Forward pass through layer ℓ.
    $\mathbf{d} \leftarrow \mathbf{W}_\epsilon^\ell \mathbf{d}$
  end for
  return λ
end function

goal of the paper is contrasting learned piece-wise affine functions, rather than optimizing networks for extreme performance.

C Linear Region Discovery Algorithm

This section presents the pseudocode for the linear region discovery algorithm introduced in section 2.3, together with a complexity analysis.

Linear Region Discovery on Compact 1-Dimensional Domains Let $\mathbf{x}_0, \mathbf{x}_1 \in \mathbb{R}^d$ be two distinct points in the input space of a ReLU network $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^K$, and let $\mathbf{d} = \mathbf{x}_1 - \mathbf{x}_0$ be the corresponding direction vector. Consider the line segment $\boldsymbol{\pi}(t) : \mathcal{I} = [0, 1] \to \mathbb{R}^d$ such that $t \mapsto \mathbf{x}_0 + t\mathbf{d}$. As observed in section 2.3, $\mathbf{f}$ implicitly induces a partition $\mathcal{P}$ of $\boldsymbol{\pi} \in \mathbb{R}^d$ into disjoint linear regions, whose entry and exit points along $\boldsymbol{\pi}$ in turn induce a partition $\mathcal{P}_\mathcal{I}$ of $\mathcal{I}$, with $\mathcal{P}_\mathcal{I} = \{t_\epsilon \in \mathcal{I} : 0 = t_0 < \ldots < t_D = 1\}$. Algorithm 2 describes a procedure for numerically determining $\mathcal{P}_\mathcal{I}$.

Starting from $\mathbf{x} = \mathbf{x}_0 \in A_0$, our method travels along $\mathbf{d}$ by computing the smallest displacement λ_0 to cross a linear region boundary of A_0 along $\mathbf{d}$. Afterward, $\mathbf{x}$ is moved to the linear region boundary, $\mathbf{x} = \mathbf{x} + \lambda_0 \mathbf{d}$, and the procedure is repeated until $\mathbf{x}$ falls in the same linear region as $\mathbf{x}_1$.

For each linear region discovered, the smallest lambda to cross one of its boundaries is computed by solving Equation 3, as detailed in Algorithm 1. For numerical stability, when solving for λ, all values below the sensitivity threshold τ are discarded.

Linear region membership of $\mathbf{x} \in A_\epsilon$ is defined by using the corresponding activation pattern $\mathbf{x}$ $\big[\text{vec}(\mathbf{M}_\mathbf{x}^1), \ldots, \text{vec}(\mathbf{M}_\mathbf{x}^L)\big]$, obtained when forward-passing $\mathbf{x}$ through $\mathbf{f}$.

Finally, the algorithm terminates under any of the following conditions. If the activation pattern of $\mathbf{x}$ is the same as the one of $\mathbf{x}_1$, then all linear regions have been discovered and the procedure completes. Furthermore, to ensure numerical stability, Algorithm 2 terminates also if no finite λ can be found, or if λ overshoots $\mathbf{x}_1$. The former condition occurs only when — either exactly or approximately — $\mathbf{d}^\ell$ is contained in all linear region boundaries, for every neuron in the network. The latter occurs when, for every neuron in the network, $\mathbf{d}$ is approximately parallel to each hyperplane. We didn't observe any such case in practice.

In section 3, we run our algorithm on data-driven trajectories obtained by connecting multiple line-paths $\boldsymbol{\pi}$ to form a closed loop in the input space. Throughout our experiments, computations are performed with double precision, with sensitivity $\tau = 1\text{e}-6$, for normalized pixel values and unnormalized direction $\mathbf{d}$[4].

Complexity Analysis A naïve implementation of algorithms 1 and 2 entails multiple sequential bottlenecks. For a set of N paths $\{\boldsymbol{\pi}_n\}_{n=1}^N$, each consisting of A line segments, estimating the respective density D_n along

[4]In the scale of normalized directions $\hat{\mathbf{d}} = \frac{\mathbf{x}_1 - \mathbf{x}_0}{\|\mathbf{x}_1 - \mathbf{x}_0\|_2}$, τ is approximately $1\text{e}-9$.

each $\boldsymbol{\pi}_n$, involves iterating through each path, line segment, layer ℓ of $\mathbf{f}$, as well as neuron in each layer, for a total run-time complexity of $\mathcal{O}(2 \cdot NALDV)$, with $D := \max_n D_n$, $V = \sum_{\ell=1}^{L} d_\ell$, and the factor of 2 arising from the need of forward-propagating both $\mathbf{x}$ and $\mathbf{d}$ in Algorithm 2. Specifically, such complexity is the same as for analytical methods, as described by Zhang & Wu (2020); Hanin & Rolnick (2019b).

In our implementation, we collate all line segments into a single set of $N \cdot A$ line paths, and estimate density in parallel over batches of $B = 1024$ lines at a time, reducing the data-dependent complexity to $\mathcal{O}(\frac{NA}{B})$. Furthermore, for each layer, Algorithm 1 allows to parallelize computation across all neurons composing the layer, bringing down the complexity to $\mathcal{O}(2\frac{NA}{B}LD)$. Additionally, in our implementation, we batch $\mathbf{x}$ and $\mathbf{d}$ together, so that each λ can be estimated by running a single forward pass [5], bringing down the complexity to $\mathcal{O}(\frac{NA}{B}LD)$. Finally, we note that sampling-based methods would require estimating D a priori, making counting either imprecise or more expensive than needed. Our algorithm exactly recovers all linear regions.

We observe that the dependency on the depth L of $\mathbf{f}$ is intrinsic in the nature of forward passes in feed-forward networks, and cannot currently be removed.

In conclusion, to quantify complexity in practice, we note that the run-time complexity of training the same network $\mathbf{f}$ on a training set of size S, using SGD with batch size B, for E epochs, involves running $\lfloor \frac{S}{B} \rfloor E$ forward passes, with total[6] complexity $\mathcal{O}(\frac{S}{B}EL)$. In practice, for a network trained on the CIFAR-10 training split, of size $S = 50000$ samples, for $E = 200$ epochs, the number of forward passes is approximately 78000. For VGG8 such number is thus similar to the average observed density D (c.f. Figure 4), while it is 4 times lower for ResNet18.

Algorithm 2 Linear region discovery on compact 1-D domains.

function FindLinearRegions($\mathbf{f}, \mathbf{x}_0, \mathbf{x}_1, \tau$)
  $\mathbf{c} \leftarrow$ ActivationPattern($\mathbf{f}, \mathbf{x}_0$)
  $\mathbf{c}_1 \leftarrow$ ActivationPattern($\mathbf{f}, \mathbf{x}_1$)
  $\mathbf{d} \leftarrow \frac{\mathbf{x}_1 - \mathbf{x}_0}{\|\mathbf{x}_1 - \mathbf{x}_0\|_2}$
  $\mathbf{x} \leftarrow \mathbf{x}_0$
  $\mathcal{P}_\mathcal{I} \leftarrow \emptyset$
  while $\mathbf{c} \neq \mathbf{c}_1$ **do**
    $\lambda \leftarrow$ FindLambda($\mathbf{f}, \mathbf{x}, \mathbf{d}, \tau$)
    $\mathcal{P}_\mathcal{I} \leftarrow \mathcal{P}_\mathcal{I} \cup \{\lambda\}$
    if $\lambda = \infty$ **then** ▷ Degenerate case. No finite lambda found.
      break
    end if
    $\mathbf{x} \leftarrow \mathbf{x} + \lambda \mathbf{d}$ ▷ Step until linear region boundary
    if $\|\mathbf{x}_0 - \mathbf{x}_1\|_2 < \|\mathbf{x} - \mathbf{x}_0\|_2$ **then** ▷ Overshot $\mathbf{x}_1$
      break
    end if
    $\mathbf{c} \leftarrow$ ActivationPattern($\mathbf{f}, \mathbf{x}$)
  end while
  return $|\mathcal{P}_\mathcal{I}|, \mathcal{P}_\mathcal{I}$ ▷ Return density, as well as all boundary points.
end function

[5] In our code this is achieved by exploiting Pytorch's forward-hooks.

[6] This is clearly a lower bound on the complexity of training, since backward passes are not accounted for. Furthermore, as is the case for training, optimizations like distributed data-parallel paradigms can be applied to our method, allowing to process larger batch sizes, thus reducing the number of forward passes required to $\mathcal{O}(LD)$.

D Computing Absolute Deviation

In this section, we present the full derivation of absolute deviation, introduced in section 2.4. Recalling equation 4,

$$\begin{aligned}\int_{\boldsymbol{\pi}} |f^k(\mathbf{x}) - a^k(\mathbf{x})| d\mathbf{x} &= \sum_{\epsilon=0}^{D-1} \int_{t_\epsilon}^{t_{\epsilon+1}} |f^k(\boldsymbol{\pi}(t)) - a^k(\boldsymbol{\pi}(t))| \cdot \|\dot{\boldsymbol{\pi}}(t)\| dt \\ &= \sum_{\epsilon=0}^{D-1} \int_{t_\epsilon}^{t_{\epsilon+1}} |f^k(\mathbf{x}_0 + t\mathbf{d}) - a^k(\mathbf{x}_0 + t\mathbf{d})| \cdot \|\mathbf{d}\| dt \\ &= \|\mathbf{d}\| \cdot \sum_{\epsilon=0}^{D-1} \int_{t_\epsilon}^{t_{\epsilon+1}} \big(|f_\epsilon^k(\mathbf{x}_0) - f_0^k(\mathbf{x}_0) + t\big(f_\epsilon^k(\mathbf{x}_1) - f_\epsilon^k(\mathbf{x}_0) - f_1^k(\mathbf{x}_1) + f_0^k(\mathbf{x}_0)\big)|\big) dt\end{aligned}$$

we note that, on each linear region A_ϵ, the affine function inside the absolute value of the integrand is not necessarily monotonic, as f^k and a^k may intersect at

$$Z = \frac{f_0^k(\mathbf{x}_0) - f_\epsilon^k(\mathbf{x}_0)}{f_\epsilon^k(\mathbf{x}_1) - f_\epsilon^k(\mathbf{x}_0) - f_1^k(\mathbf{x}_1) + f_0^k(\mathbf{x}_0)} \tag{5}$$

If this is the case, then the integral over A_ϵ can be split as the sum of integrals over two sub-intervals $[t_\epsilon, t_*] \cup [t_*, t_{\epsilon+1}]$, yielding:

$$\begin{aligned}\int_{\boldsymbol{\pi}} |f^k(\mathbf{x}) - a^k(\mathbf{x})| d\mathbf{x} =& \|\mathbf{d}\| \cdot \sum_{\epsilon=0}^{D-1} \int_{t_\epsilon}^{t_*} |f_\epsilon^k(\mathbf{x}_0) - f_0^k(\mathbf{x}_0) + t\big(f_\epsilon^k(\mathbf{x}_1) - f_\epsilon^k(\mathbf{x}_0) - f_1^k(\mathbf{x}_1) + f_0^k(\mathbf{x}_0)\big)| dt + \\ &+ \|\mathbf{d}\| \cdot \sum_{\epsilon=0}^{D-1} \int_{t_*}^{t_{\epsilon+1}} |f_\epsilon^k(\mathbf{x}_0) - f_0^k(\mathbf{x}_0) + t\big(f_\epsilon^k(\mathbf{x}_1) - f_\epsilon^k(\mathbf{x}_0) - f_1^k(\mathbf{x}_1) + f_0^k(\mathbf{x}_0)\big)| dt\end{aligned} \tag{6}$$

Otherwise — if the intersection between f^k and a^k occurs outside of A_ϵ — we take t_* as $t_* = \max\{t_\epsilon, \min\{Z, t_{\epsilon+1}\}\}$, which can be efficiently solved as a simple linear programme.

Finally, for each $k = 1, \ldots, K$, absolute deviation is given by

$$\begin{aligned}\int_{\boldsymbol{\pi}} |f^k(\mathbf{x}) - a^k(\mathbf{x})| d\mathbf{x} =& \|\mathbf{d}\| \Big((t_* - t_\epsilon)|f_\epsilon^k(\mathbf{x}_0) - f_0^k(\mathbf{x}_0)| + (t_{\epsilon+1} - t_*)|f_\epsilon^k(\mathbf{x}_0) - f_0^k(\mathbf{x}_0)| \quad + \\ & \frac{t_*^2 - t_\epsilon^2}{2} |f_\epsilon^k(\mathbf{x}_1) - f_\epsilon^k(\mathbf{x}_0) - f_1^k(\mathbf{x}_1) + f_0^k(\mathbf{x}_0)| \quad + \\ & \frac{t_{\epsilon+1}^2 - t_*^2}{2} |f_\epsilon^k(\mathbf{x}_1) - f_\epsilon^k(\mathbf{x}_0) - f_1^k(\mathbf{x}_1) + f_0^k(\mathbf{x}_0)| \Big)\end{aligned} \tag{7}$$

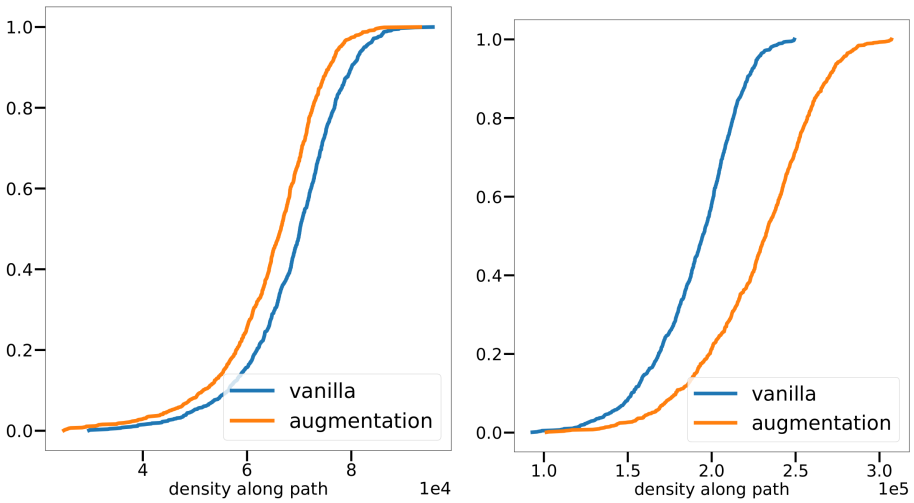
E Data-Driven Trajectories

Each data-driven trajectory $\boldsymbol{\pi}_n$ in section 3 is obtained from a starting sample $\mathbf{x}_n^0$ and A augmented versions $\{\mathbf{x}_n^a : a = 1, \ldots, A-1\}$, so that $\boldsymbol{\pi}_n$ is composed of line paths connecting each $\mathbf{x}_n^a$ with $\mathbf{x}_n^{a+1\%A}$, for $a = 0, \ldots, A-1$, forming a closed loop.

Each augmented image $\mathbf{x}_n^a$ is obtained by translating the starting point $\mathbf{x}_n^0$ along a circular trajectory of radius $r = 4$, with translation parameter $s = (r\cos\alpha_a, r\sin\alpha_a)$, for $\alpha_a = a \cdot \frac{2\pi}{A}$. Specifically, $\mathbf{x}_n^0$ is first padded by 2 pixels on each edge, then translated along the circular trajectory, and finally cropped back to its original resolution. Finally, in addition to Novak et al. (2018), we ensure that no edge artifacts are introduced in the augmented images in the transformation process.

F Additional Experiments

Ablating Data-Driven Trajectories In this section, we consider alternatives to estimating density and absolute deviation along closed loops in the input space. We begin by recalling that, since both density and



(a) (b)

Figure 7: ECDF of density along data-driven paths on CIFAR-10 samples, for a network trained with data augmentation and without ("vanilla"). a) VGG8. b) ResNet18.

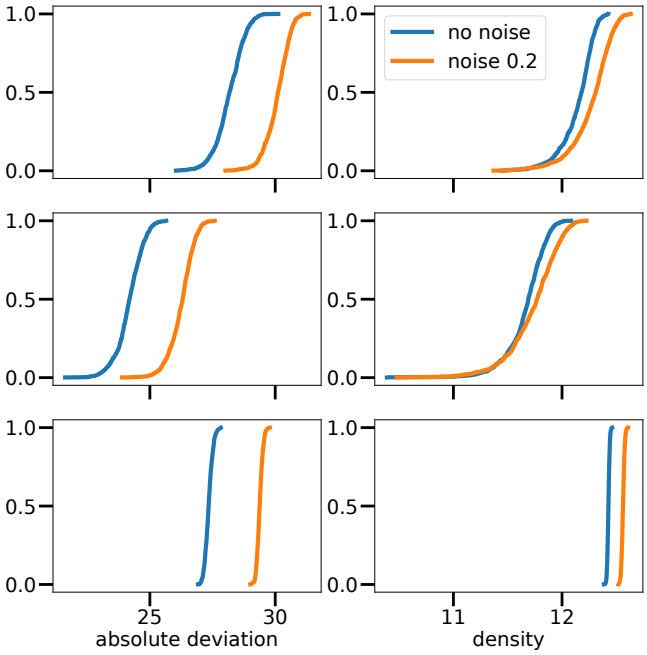


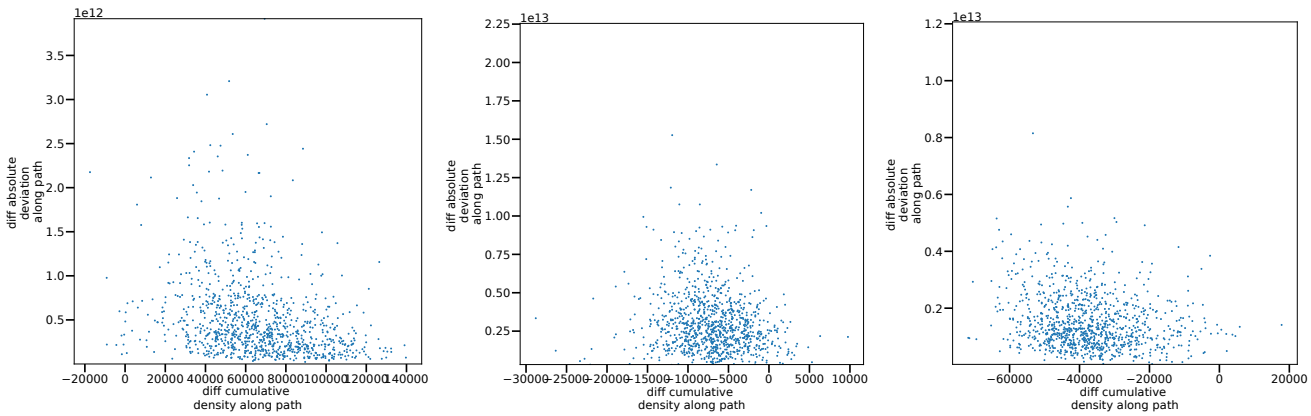
Figure 8: (Log x-scale) Absolute deviation and density for circular trajectories (top); open paths connecting weak augmentations of a base train image (middle); circular paths connecting uniform noise with per-pixel statistics as the CIFAR-10 training split (bottom).

absolute deviation are computed as line integrals on compact 1D supports, alternative strategies, like MC-based sampling of random directions in pixel space, increase chances of double counting linear regions, making computations more memory intensive. Hence, 1-D paths are a core contribution w.r.t. scalability of our method. Figure 8 presents ablations on circular paths, in which three alternatives are considered, namely closed loops as in the main experiments (top), open paths connecting weak augmentations (1-pixel shifts) of a base training image (middle), and finally closed paths connecting uniform random noise with the same per-pixel statistics as the CIFAR-10 training split (bottom).

Under all settings, absolute deviation correctly separates ResNet18s respectively trained on clean as well as 20% noisy labels on CIFAR-10, while density struggles on trajectories that lie close to the train data. Finally, both density and deviation are almost uniformly distributed if measured away from the training data, corroborating in the setting of modern networks what observed by Novak et al. (2018).

Distribution of Density for Networks Trained with Data Augmentation Extending Figure 4, we study the distribution of density values for VGG8 and ResNet18 trained with and without data augmentation, reporting our findings in Figure 7. It can be observed that, while density correctly separates VGG8 trained with and without data augmentation, the distance between the two cumulative distribution functions is less marked, in contrast with absolute deviation in Figure 3, which more strongly distinguishes between smoothness induced by data augmentation, and vanilla training. Finally, the ECDF curves are ranked in the opposite order for ResNet18 (Figure 7, right), which produces more regions when data augmentation is enabled. This shows that, unike absolute deviation, density is an unreliable predictor of nonlinearity for trained networks, producing inconsistent results when different architectures are compared.

Distribution of Paired Differences Complementing Table 2, this section provides a few representative examples of the distribution of instance-level differences, showing how density and absolute deviation change on individual paths when the training settings change. Figure 9 shows the distribution of density and absolute deviation for three settings. First, when comparing a trained ResNet18 on CIFAR-10 with its initialization (left), density can meaningfully capture increased nonlinearity, as showed by the distribution of differences being shifted towards positive values (0.99 ± 0.01 positive differences for density). Second, for a VGG8 trained with and without data augmentation on CIFAR-100 (middle), we see how density becomes a noisy estimator of nonlinearity, as indicated by the distribution of differences being shifted towards negative values (0.38 ± 0.24 positive differences for density). Third, for a ResNet18 trained with and without augmentation on CIFAR-10 (right), density fails to capture the difference in nonlinearity, with the distribution of differences being strongly biased towards negative values (0.03 ± 0.03 positive differences for density). In contrast, absolute deviation consistently and more reliably detects increased nonlinearity at the level of individual paths (as shown by always positive distribution values), thus better capturing the nonlinear behaviour of piece-wise affine functions for ReLU networks.



(a) (b) (c)

Figure 9: Distribution of per-path paired differences of density (x-axis) and absolute deviation (y-axis). (Left) Paired differences for a ResNet18 trained with data augmentation on CIFAR-10 vs the same network at initialization. Here, both density and absolute deviation are able to strongly detecting increased nonlinearity at the level of individual paths. (Middle) Paired differences for a VGG8 trained on CIFAR-100 with and without data augmentation. While absolute deviation captures the regularity induced by data augmentation, density becomes a noisy estimator of nonlinearity. (Right) Paired differences for a ResNet18 trained on CIFAR-10 with and without data augmentation. Similarly to VGG8/CIFAR-100, density fails to measure increased nonlinearity when data augmentation is disabled.