Efficient Training of Low-Curvature Neural Networks

Suraj Srinivas[*1]
Harvard University
ssrinivas@seas.harvard.edu

Kyle Matoba[*]
Idiap Research Institute & EPFL
kyle.matoba@epfl.ch

Himabindu Lakkaraju
Harvard University
hlakkaraju@hbs.edu

François Fleuret
University of Geneva
francois.fleuret@unige.ch

Abstract

Standard deep neural networks often have excess non-linearity, making them susceptible to issues such as low adversarial robustness and gradient instability. Common methods to address these downstream issues, such as adversarial training, are expensive and often sacrifice predictive accuracy.

In this work, we address the core issue of excess non-linearity via curvature, and demonstrate low-curvature neural networks (LCNNs) that obtain drastically lower curvature than standard models while exhibiting similar predictive performance. This leads to improved robustness and stable gradients, at a fraction of the cost of standard adversarial training. To achieve this, we decompose overall model curvature in terms of curvatures and slopes of its constituent layers. To enable efficient curvature minimization of constituent layers, we introduce two novel architectural components: first, a non-linearity called centered-softplus that is a stable variant of the softplus non-linearity, and second, a Lipschitz-constrained batch normalization layer.

Our experiments show that LCNNs have lower curvature, more stable gradients and increased off-the-shelf adversarial robustness when compared to standard neural networks, all without affecting predictive performance. Our approach is easy to use and can be readily incorporated into existing neural network architectures.

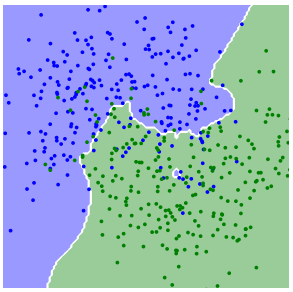
Code to implement our method and replicate our experiments is available at https://github.com/kylematoba/lcnn.

1 Introduction

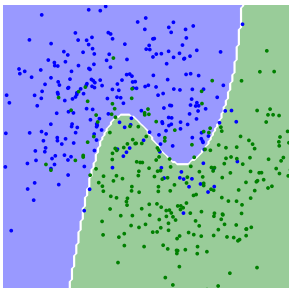
The high degree of flexibility present in deep neural networks is critical to achieving good performance in complex tasks such as image classification, language modelling and generative modelling of images [1–3]. However, having excessive flexibility is undesirable as this can lead to model under-specification [4] which results in unpredictable behaviour on out-of-domain inputs, such as vulnerability to adversarial examples. Such under-specification can be avoided in principle via Occam's razor, which requires training models that are as simple as possible for the task at hand, and not any more. Motivated by this principle, in this work we focus on training neural network models without excess non-linearity (e.g.: see Fig 1b), such that predictive performance remains unaffected.

[*]Equal Contribution
[1]Work done partially at Idiap Research Institute

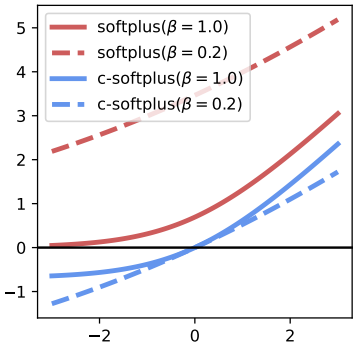
36th Conference on Neural Information Processing Systems (NeurIPS 2022).



(a) Decision boundary with standard NN ($\beta = 1000$)



(b) Decision boundary with LCNN ($\beta = 1$)



(c) Softplus vs Centered-Softplus

Figure 1: Decision boundaries of (a) standard NN and (b) LCNN trained on the two moons dataset. LCNN recovers highly regular decision boundaries in contrast to the standard NN. (c) Comparison of softplus and centered-softplus non-linearities (defined in §3.3). These behave similarly for large β values, and converge to linear maps for low β values. However, softplus diverges while centered-softplus stays close to the origin.

Central to this work is a precise notion of curvature, which is a mathematical quantity that encodes the flexibility or the degree of non-linearity of a function at a point. In deep learning, the curvature of a function at a point is often quantified as the norm of the Hessian at that point [5–7]. Hessian norms are zero everywhere if and only if the function is linear, making them suitable to measure the degree of non-linearity. However, they suffer from a dependence on the scaling of model gradients, which makes them unsuitable to study its interplay with model robustness. In particular, robust models have small gradient norms [8], which naturally imply smaller Hessian norms. But are they truly more linear as a result? To be able to study robustness independent of non-linearity, we propose to use *normalized curvature*, which normalizes the Hessian norm by its corresponding gradient norm, thus disentangling the two measures. Surprisingly, we find that normalized curvature is a stable measure across train and test samples (see Table 3), whereas the usual curvature is not.

A straightforward approach to train low-curvature models [6, 9] involves directly penalizing curvature locally at training samples. However, these are inefficient as they involve expensive Hessian computations, and only minimize local point-wise curvature and not curvature everywhere. A complementary approach is that of Dombrowski et al. [7], who propose architectures that have small global curvature, but do not explicitly penalize this curvature during training. In contrast, we propose efficient mechanisms to explicitly penalize the *normalized curvature* globally. In addition, while previous methods [6, 7] penalize the Frobenius norm of the Hessian, we penalize its spectral norm, thus providing tighter and more interpretable robustness bounds.[1]

Our overall contributions in this paper are:

1. In §3.1 we propose to measure curvature of deep models via *normalized curvature*, which is invariant to scaling of the model gradients.
2. In §3.2 we show that normalized curvature of neural networks can be upper bounded via a decomposition in terms of normalized curvatures and slopes of individual layers.
3. We introduce architectural innovations including a novel activation function called the centered-softplus (§3.3), and a novel Lipschitz constrained batch normalization layer (§3.4.2).
4. In §4 we prove bounds on the relative gradient robustness and adversarial robustness of models in terms of normalized curvature, showing that controlling normalized curvature directly controls these properties.
5. In §5, we show experiments demonstrating that our proposed innovations are successful in training low-curvature models without sacrificing training accuracy, and that such models have robust gradients, and are more robust to adversarial examples out-of-the-box.

[1]Note that the Frobenius norm of a matrix strictly upper bounds its spectral norm

2 Related Work

Adversarial Robustness of Neural Networks: The well-known phenemenon of adversarial vulnerabilities of neural networks [10, 11] shows that adding small amounts of imperceptible noise can cause deep neural networks to misclassify points with high confidence. The canonical method to defend against this vulnerability is adversarial training [12] which trains models to accurately classify adversarial examples generated via an attack such as projected gradient descent (PGD). However, this approach is computationally expensive and provides no formal guarantees on robustness. Cohen et al. [13] proposed randomized smoothing, which provides a formal guarantee on robustness by generating a smooth classifer from any black-box classifier. Hein and Andriushchenko [8] identified the local Lipschitz constant as critical quantity to prove formal robustness guarantees. Moosavi Dezfooli et al. [6] penalize the Frobenius norm of the Hessian, and show that they performs similarly to models trained via adversarial training. Qin et al. [9] introduce a local linearity regularizer, which also implicitly penalizes the Hessian. Similar to these works, we enforce low curvature to induce robustness, but we focus on out of the box robustness of LCNNs.

Unreliable Gradient Interpretations in Neural Networks: Gradient explanations in neural networks can be unreliable. Ghorbani et al. [14], Zhang et al. [15] showed that for any input, it is possible to find adversarial inputs such that the gradient explanations for these points that are highly dissimilar to each other. Srinivas and Fleuret [16] showed that pre-softmax logit gradients however are independent of model behaviour, and as a result we focus on post-softmax loss gradients in this work. Ros and Doshi-Velez [17] showed empirically that robustness can be improved by gradient regularization, however Dombrowski et al. [18] showed that gradient instability is primarily due to large Hessian norms. This suggests that the gradient penalization in Ros and Doshi-Velez [17] performed unintentional Hessian regularization, which is consistent with our experimental results. To alleviate this, Dombrowski et al. [7] proposed to train low curvature models via softplus activations and weight decay, broadly similar to our strategy. However, while Dombrowski et al. [7] focused on the Frobenius norm of the Hessian, we penalize the normalized curvature – a scaled version of the Hessian spectral norm – which is strictly smaller than the Frobenius norm, and this results in a more sophisticated penalization strategy.

Lipschitz Layers in Neural Networks There has been extensive work on methods to bound the Lipschitz constant of neural networks. Cisse et al. [19] introduced Parseval networks, which penalizes the deviation of linear layers from orthogonality, since an orthogonal linear operator evidently has a Lipschitz constant of one, this shrinks the Lipschitz constant of a layer towards one. Trockman and Kolter [20] use a reparameterization of the weight matrix, called the Cayley Transform, that is orthogonal by construction. Miyato et al. [21], Ryu et al. [22] proposed spectral normalization, where linear layers are re-parameterized by dividing by their spectral norm, ensuring that the overall spectral norm of the parameterized layer is one.

3 Training Low-Curvature Neural Networks

In this section, we introduce our approach for training low-curvature neural nets (LCNNs). Unless otherwise specified, we shall consider a neural network classifier f that maps inputs $\mathbf{x} \in \mathbb{R}^d$ to a scalar loss value $f(\mathbf{x}) \in \mathbb{R}^+$.

3.1 Measuring Relative Model Non-Linearity via Normalized Curvature

We begin our analysis by discussing a desirable definition of curvature $\mathcal{C}_f(\mathbf{x}) \in \mathbb{R}^+$. While curvature is well-studied topic in differential geometry [23] where Hessian normalization is a common theme, our discussion will be largely independent of this literature, in a manner more suited to requirements in deep learning. Regardless of the literature, a typical property of a curvature measure is that $\mathcal{C}_f(\mathbf{x}) = 0 \;\; \forall\, \mathbf{x} \in \mathbb{R}^d \iff f$ is linear, and hence $\max_{\mathbf{x}\in\mathbb{R}^d} \mathcal{C}_f(\mathbf{x})$ can be thought of as a measure of a model's non-linearity. A common way to define curvature in machine learning [5–7] has been via Hessian norms. However, these measures are sensitive to gradient scaling, which is undesirable. After all, the degree of model non-linearity must intuitively be independent of how large its gradients are.

For example, if two functions f, g are scaled ($f = k \times g$), or rotated versions of each other ($\nabla f = k \times \nabla g$), then we would like them to have similar curvatures in the interest of disentangling curvature

(i.e., degree of non-linearity) from scaling. It is easy to see that Hessian norms $\|\nabla^2_{\mathbf{x}} f(\mathbf{x})\|_2$ do not have this property, as scaling the function also scales the corresponding Hessian. We would like to be able to recover low-curvature models with steep gradients (that are non-robust), precisely to be able to disentangle their properties from low-curvature models with small gradients (that are robust), which Hessian norms do not allow us to do.

To avoid this problem, we propose a definition of curvature that is approximately *normalized*: $\mathcal{C}_f(\mathbf{x}) = \|\nabla^2 f(\mathbf{x})\|_2 / (\|\nabla f(\mathbf{x})\|_2 + \varepsilon)$. Here $\|\nabla f(\mathbf{x})\|_2$ and $\|\nabla^2 f(\mathbf{x})\|_2$ are the ℓ_2 norm of the gradient and the spectral norm of the Hessian, respectively, where $\nabla f(\mathbf{x}) \in \mathbb{R}^d, \nabla^2 f(\mathbf{x}) \in \mathbb{R}^{d \times d}$, and $\varepsilon > 0$ is a small constant to ensure well-behavedness of the measure. This definition measures Hessian norm *relative* to the gradient norm, and captures a notion of relative local linearity, which can be seen via a straightforward application of Taylor's theorem:

$$\underbrace{\frac{\|f(\mathbf{x}+\epsilon) - f(\mathbf{x}) - \nabla f(\mathbf{x})^\top \epsilon\|_2}{\|\nabla f(\mathbf{x})\|_2}}_{\text{relative local linearity}} \leq \frac{1}{2} \underbrace{\max_{\mathbf{x} \in \mathbb{R}^d} \mathcal{C}_f(\mathbf{x})}_{\text{normalized curvature}} \|\epsilon\|_2. \tag{1}$$

Here the numerator on the left captures the local linearity error, scaled by the gradient norm in the denominator, which can be shown using Taylor's theorem to be upper bounded by the normalized curvature. We shall consider penalizing this notion of normalized curvature, and we shall simply refer to this quantity as curvature in the rest of the paper.

3.2 A Data-Free Upper Bound on Curvature

Directly penalizing the curvature is computationally expensive as it requires backpropagating an estimate of the Hessian norm, which itself requires backpropagating gradient-vector products. This requires chaining the entire computational graph of the model at least three times. Moosavi Dezfooli et al. [6] reduce the complexity of this operation by computing a finite-difference approximation to the Hessian from gradients, but even this double-backpropagation is expensive. We require an efficient penalization procedure takes a single backpropagation step, similar to ordinary neural network training.

To this end, we propose to minimize a data-free upper bound on the curvature. To illustrate the idea, we first show this upper bound for the simplified case of the composition of one-dimensional functions ($f : \mathbb{R} \to \mathbb{R}$).

Lemma 1. *Given a 1-dimensional compositional function $f = f_L \circ f_{L-1} \circ ... \circ f_1$ with $f_i : \mathbb{R} \to \mathbb{R}$ for $i = 1, 2, \ldots, L$, the normalized curvature $\mathcal{C}_f := |f''/f'|$, is bounded by $\left|\frac{f''}{f'}\right| \leq \sum_{i=1}^{L} \left|\frac{f_i''}{f_i'}\right| \prod_{j=1}^{i} |f_j'|$*

Proof. We first have $f' = \prod_{i=1}^{L} f_i'$. Differentiating this expression we have $f'' = \sum_{i=1}^{L} f_i'' \prod_{j=1}^{i} f_j' \prod_{k=1, k \neq i}^{L} f_k'$. Dividing by f', taking absolute value of both sides, and using the triangle inequality, we have the intended result.

Extending the above expression to functions $\mathbb{R}^n \to \mathbb{R}$ is not straightforward, as the intermediate layer Hessians are order three tensors. We derive this using a result from Wang et al. [24] that connects the spectral norms of order-n tensors to the spectral norm of their matrix "unfoldings". The full derivation is presented in the appendix, and the (simplified) result is stated below.

Theorem 1. *Given a function $f = f_L \circ f_{L-1} \circ \ldots \circ f_1$ with $f_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$, the curvature $\mathcal{C}_f$ can be bounded by the sum of curvatures of individual layers $\mathcal{C}_{f_i}(\mathbf{x})$, i.e.,*

$$\mathcal{C}_f(\mathbf{x}) \leq \sum_{i=1}^{L} \mathcal{C}_{f_i}(\mathbf{x}) \prod_{j=1}^{i} \|\nabla_{f_{j-1}} f_j(\mathbf{x})\|_2 \leq \sum_{i=1}^{L} n_i \times \max_{\mathbf{x}} \mathcal{C}_{f_i}(\mathbf{x}) \prod_{j=1}^{i} \max_{\mathbf{x}} \|\nabla_{f_{j-1}} f_j(\mathbf{x})\|_2. \tag{2}$$

The rightmost term is independent of $\mathbf{x}$ and thus holds uniformly across all data points. This bound shows that controlling the curvature and Lipschitz constant of each layer of a neural network enables us to control the overall curvature of the model.

Practical neural networks typically consist of[1] linear maps such as convolutions, fully connected layers, and batch normalization layers, and non-linear activation functions. Linear maps have zero curvature by definition, and non-linear layers often have bounded gradients ($= 1$), which simplifies computations. In the sections that follow, we shall see how to penalize the remaining terms, i.e., the curvature of the non-linear activations, and the Lipschitz constant of the linear layers.

3.3 Centered-Softplus: Activation Function with Trainable Curvature

Theorem 2 shows that the curvature of a neural network depends on the curvature of its constituent activation functions. We thus propose to use activation functions with minimal curvature.

The kink at the origin of the ReLU function implies an undefined second derivative. On the other hand, a smooth activation such as the softplus function, $s(x;\beta) = \log(1 + \exp(\beta x))/\beta$ is better suited to analyzing questions of curvature. Despite not being a common baseline choice, softplus does see regular use, especially where its smoothness facilitates analysis [25]. The curvature of the softplus function is

$$\mathcal{C}_{s(\cdot;\beta)}(x) = \beta \times \left(1 - \frac{\mathrm{d}s(x;\beta)}{\mathrm{d}x}\right) \leq \beta \tag{3}$$

Thus using softplus with small β values ensures low curvature. However, we observe two critical drawbacks of softplus preventing its usage with small β: (1) divergence for small β, where $s(x;\beta \to 0) = \infty$ which ensures that well-behaved low curvature maps cannot be recovered and, (2) instability upon composition i.e., $s^n(x = 0;\beta) = \underbrace{s \circ s \circ ... \circ s}_{n \text{ times}}(x = 0;\beta) = \frac{\log(n+1)}{\beta}$. Thus we have that $s^n(x = 0;\beta) \to \infty$ as $n \to \infty$, which shows that composing softplus functions exacerbates instability around the origin. This is critical for deep models with large number of layers n, which is precisely the scenario of interest to us. To remedy this problem, we propose *centered-softplus* $s_0(x;\beta)$, a simple modification to softplus by introducing a normalizing term as follows.

$$s_0(x;\beta) = s(x;\beta) - \frac{\log 2}{\beta} = \frac{1}{\beta}\log\left(\frac{1 + \exp(\beta x)}{2}\right). \tag{4}$$

This ensures that $s(x = 0;\beta) = s^n(x = 0;\beta) = 0$ for any positive integer n, and hence also ensures stability upon composition. More importantly, we have $s_0(x;\beta \to 0) = x/2$ which is a scaled linear map, while still retaining $s_0(x;\beta \to \infty) = \mathrm{ReLU}(x)$. This ensures that we are able to learn both well-behaved linear maps, as well as highly non-linear ReLU-like maps if required.

We further propose to cast β is a learnable parameter and penalize its value, hence directly penalizing the curvature of that layer. Having accounted for the curvature of the non-linearities, the next section discusses controlling the gradients of the linear layers.

3.4 Lipschitz Linear Layers

Theorem 2 shows that penalizing the Lipschitz constants of the constituent linear layers in a model is necessary to penalize overall model curvature. There are broadly three classes of linear layers we consider: convolutions, fully connected layers, and batch normalization.

3.4.1 Spectrally Normalized Convolutions and Fully Connected Layers

We propose to penalize the Lipschitz constant of convolutions and fully connected layer via existing spectral normalization-like techniques. For fully connected layers, we use vanilla spectral normalization Miyato et al. [21] which controls the spectral norm of a fully connected layer by reparameterizing – replace a weight matrix W, by $W/||W||_2$ which has a spectral norm of unity. Ryu et al. [22] generalize this to convolutions by presenting a power iteration method that works directly on the linear mapping implicit in a convolution: maintaining 3D left and right singular "vectors" of the 4D tensor of convolutional filters and developing the corresponding update rules. They call this

[1] We ignore self-attention layers in this work.

"real" spectral normalization to distinguish it from the approximation that [21] proposed for convolutions. Using spectral normalization on fully connected layers and "real" spectral normalization on convolutional layers ensures that the spectral norm of these layers is exactly equal to one, further simplifying the bound in Theorem 2.

3.4.2 γ-Lipschitz Batch Normalization

In principle, at inference time batch normalization (BN) is multiplication by the diagonal matrix of inverse running standard deviation estimates.[1]. Thus we can spectrally normalize the BN layer by computing the reciprocal of the smallest running standard deviation value across dimensions i, i.e., $||\text{BN}||_2 = \max_x ||\text{BN}(x)||_2 = 1/\min_i \text{running-std}(i)$. In practice, we found that models with such spectrally normalized BN layers tended to either diverge or fail to train in the first place, indicating that the scale introduced by BN is necessary for training. To remedy this, we introduce the γ-*Lipschitz* batch normalization, defined by

$$\begin{aligned}
\text{1-Lipschitz-BN}(\mathbf{x}) &\leftarrow \text{BN}(\mathbf{x})/||\text{BN}||_2 \\
\gamma\text{-Lipschitz-BN}(\mathbf{x}) &\leftarrow \underbrace{\min(\gamma, ||\text{BN}||_2)}_{\text{scaling factor}\leq\gamma} \times \text{1-Lipschitz-BN}(\mathbf{x}).
\end{aligned}$$

By clipping the scaling above at γ (equivalently, the running standard deviation below, at $1/\gamma$), we can ensure that the Lipschitz constant of a batch normalization layer is at most equal to $\gamma \in \mathbb{R}^+$. We provide a PyTorch-style code snippet in the appendix. As with β, described in §3.3, we cast γ as a learnable parameter in order to penalize it during training. Unknown to us, Gouk et al. [26] proposed a similar solution, except they fit a common γ to all BN layers, whereas we fit a different γ to each layer resulting in greater flexibility.

3.5 Penalizing Curvature

We have discussed three architectural innovations – centered-softplus activations with a trainable β, spectrally normalized linear and convolution layers and a γ-Lipschitz batch normalization layer. We now discuss methods to penalize the overall curvature of a model built with these layers.

Since convolutional and fully connected layers have spectral norms equal to one by construction, they contribute nothing to curvature. Thus, we restrict attention to batch normalization and activation layers. The set of which will be subsequently referred to respectively βSP and γBN. For models with batch normalization, naively using the upper bound in Theorem 2 is problematic due to the exponential growth in the product of Lipschitz constants of batch normalization layers. To alleviate this, we propose to use a penalization $\mathcal{R}_f$ where γ_i terms are aggregated additively across batch normalization layers, and independent of the β_i terms in the following manner:

$$\mathcal{R}_f = \lambda_\beta \times \sum_{i \in \beta\text{SP}} \beta_i + \lambda_\gamma \times \sum_{j \in \gamma\text{BN}} \log \gamma_j \tag{5}$$

An additive aggregation ensures that the penalization is well-behaved during training and does not grow exponentially. Note that the underlying model is necessarily linear if the penalization term is zero, thus making it an appropriate measure of model non-linearity. Also note that $\beta_i \geq 0, \gamma_i \geq 1$ by construction. We shall henceforth use the term "LCNN" to refer to a model trained with the proposed architectural components (centered-softplus, spectral normalization, and γ-Lipschitz batch normalization) and associated regularization terms on β, γ. We next discuss the robustness and interpretability benefits that we obtain with LCNNs.

4 Why Train Low-Curvature Models?

In this section we discuss the advantages that low-curvature models offer, particularly as it pertains to robustness and gradient stability. These statements apply not just to LCNNs, but low-curvature models in general obtained via any other mechanism.

[1] We ignore the learnable parameters of batch normalization here for simplicity.

4.1 Low Curvature Models have Stable Gradients

Recent work [14, 15] has shown that gradient explanations are manipulable, and that we can easily find inputs whose explanations differ maximally from those at the original inputs, making them unreliable in practice for identifying important features. As we shall show, this amounts to models having a large curvature. In particular, we show that the relative gradient difference across points $\|\epsilon\|_2$ away in the ℓ_2 sense is upper bounded by the normalized curvature $\mathcal{C}_f$, as given below.

Proposition 1. *Consider a model f with $\max_{\mathbf{x}} \mathcal{C}_f(\mathbf{x}) \leq \delta_{\mathcal{C}}$, and two inputs $\mathbf{x}$ and $\mathbf{x}+\epsilon$ ($\in \mathbb{R}^d$). The relative distance between gradients at these points is bounded by*

$$\frac{\|\nabla f(\mathbf{x}+\epsilon) - \nabla f(\mathbf{x})\|_2}{\|\nabla f(\mathbf{x})\|_2} \leq \|\epsilon\|_2 \delta_{\mathcal{C}} \exp(\|\epsilon\|_2 \delta_{\mathcal{C}}) \sim \|\epsilon\|_2 \mathcal{C}_f(\mathbf{x}) \quad \textit{(Quadratic Approximation)}$$

The proof expands $f(\mathbf{x})$ in a Taylor expansion around $\mathbf{x}+\epsilon$ and bounds the magnitude of the second and higher order terms over the neighborhood of $\mathbf{x}$, and the full argument is given in the appendix. The upper bound is considerably simpler when we assume that the function locally quadratic, which corresponds to the rightmost term $\|\epsilon\|_2 \mathcal{C}_f(\mathbf{x})$. Thus the smaller the model curvature, the more locally stable are the gradients.

4.2 Low Curvature is Necessary for ℓ_2 Robustness

Having a small gradient norm is known to be an important aspect of adversarial robustness [8]. However, small gradients alone are not sufficient, and low curvature is necessary, to achieve robustness. This is easy to see intuitively - a model may have low gradients at a point leading to robustness for small noise values, but if the curvature is large, then gradient norms at neighboring points can quickly increase, leading to misclassification for even slightly larger noise levels. This effect is an instance of *gradient-masking* [27] , which provides an illusion of robustness by making models only locally robust.

In the result below, we formalize this intuition and establish an upper bound on the distance between two nearby points, which we show depends on both the gradient norm (as was known previously) and well as the max curvature of the underlying model within an ϵ-ball.

Proposition 2. *Consider a model f with $\max_{\mathbf{x}} \mathcal{C}_f(\mathbf{x}) \leq \delta_{\mathcal{C}}$, then for two inputs $\mathbf{x}$ and $\mathbf{x}+\epsilon$ ($\in \mathbb{R}^d$), we have the following expression for robustness*

$$\begin{aligned}\|f(\mathbf{x}+\epsilon) - f(\mathbf{x})\|_2 &\leq \|\epsilon\|_2 \|\nabla f(\mathbf{x})\|_2 \left(1 + \|\epsilon\|_2 \delta_{\mathcal{C}} \exp(\|\epsilon\|_2 \delta_{\mathcal{C}})\right) \\ &\sim \|\epsilon\|_2 \|\nabla f(\mathbf{x})\|_2 \left(1 + \|\epsilon\|_2 \mathcal{C}_f(\mathbf{x})\right) \quad (\textit{Quadratic Approximation})\end{aligned}$$

The proof uses similar techniques to those of proposition 1, and is also given in the appendix. This result shows that given two models with equally small gradients at data points, the greater robustness will be achieved by the model with the smaller curvature.

5 Experiments

In this section we perform experiments to (1) evaluate the effectiveness of our proposed method in training models with low curvature as originally intended, (2) evaluate whether low curvature models have robust gradients in practice, and (3) evaluate the effectiveness of low-curvature models for adversarial robustness. Our experiments are primarily conducted on a base ResNet-18 architecture ([28]) using the CIFAR10 and CIFAR100 datasets ([29]), and using the Pytorch [30] framework. Our methods entailed fairly modest computation – our most involved computations can be completed in under three GPU days, and all experimental results could be computed in less than 60 GPU-days. We used a mixture of GPUs – primarily NVIDIA GeForce GTX 1080 Tis – on an internal compute cluster.

Baselines and Our Methods Our baseline model for comparison involves using a ResNet-18 model with softplus activation with a high $\beta = 10^3$ to mimic ReLU, and yet have well-defined curvatures.

Another baseline is gradient norm regularization (which we henceforth call 'GradReg') [31], where the same baseline model is trained with an additional penalty on the gradient norm. We train two variants of our approach - a base LCNN, which involves penalizing the curvature, and another variant combining LCNN penalty and gradient norm regularization (LCNN + GradReg), which controls both the curvature and gradient norm. Our theory indicates that the LCNN + GradReg variant is likely to produce more robust models, which we verify experimentally. We also compare with CURE [6], softplus with weight decay [7] and adversarial training with ℓ_2 PGD [12] with noise magnitude of 0.1 and 3 iterations of PGD. We provide experimental details in the appendix.

Parameter Settings All our models are trained for 200 epochs with an SGD + momentum optimizer, with a momentum of 0.9 and an initial learning rate of 0.1 which decays by a factor of 10 at 150 and 175 epochs, and a weight decay of 5×10^{-4}.

5.1 Evaluating the Efficacy of Curvature Penalization

In this section, we evaluate whether LCNNs indeed reduce model curvature in practice. Table 1 contains our results, from which we make the following observations: (1) most of our baselines except CURE and adversarial training do not meaningfully lose predictive performance (2) GradReg and adversarially trained models are best at reducing gradient norm while LCNN-based models are best at penalizing curvature. Overall, these experimental results show that LCNN-based models indeed minimize curvature as intended.

We also observe in Table 1 that GradReg [31] has an unexpected regularizing effect on the Hessian and curvature. We conjecture that this is partly due to the following decomposition of the loss Hessian, which can be written as $\nabla^2 f(\mathbf{x}) \sim \nabla f_l(\mathbf{x}) \nabla^2_{f_l} \text{LSE}(\mathbf{x}) \nabla f_l(\mathbf{x})^\top + \nabla_{f_l} \text{LSE}(\mathbf{x}) \nabla^2 f_l(\mathbf{x})$, where $\text{LSE}(\mathbf{x})$ is the LogSumExp function, and $f_l(\mathbf{x})$ is the pre-softmax logit output. We observe that the first term strongly depends on the gradients, which may explain the Hessian regularization effects of GradReg, while the second term depends on the Hessian of the bulk of the neural network, which is penalized by the LCNN penalties. This also explains why combining both penalizations (LCNN + GradReg) further reduces curvature.

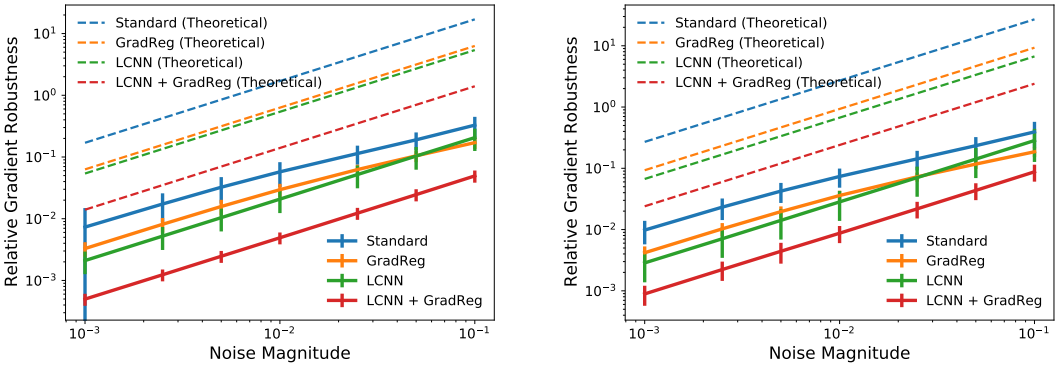
We also measure the average per-epoch training times on a GTX 1080Ti, which are: standard models / softplus + weight decay (~ 100 sec), LCNN (~ 160 sec), GradReg (~ 270 sec), LCNN+GradReg (~ 350 sec), CURE / Adversarial Training (~ 500 sec). Note that the increase in computation for LCNN is primarily due to the use of spectral normalization layers. In the The results show that LCNNs are indeed able to penalize curvature by only marginally ($1.6\times$) increasing training time, and using LCNN+GradReg only increases time $1.3\times$ over GradReg while providing curvature benefits.

Table 1: Model geometry of various ResNet-18 models trained with various regularizers on the CIFAR100 test dataset. Gradient norm regularized models [31] ('GradReg') are best at reducing gradient norms, while LCNN-based models are best at reducing curvature, leaving gradients unpenalized. We obtain the benefits of both by combining these penalties. Results are averaged across two runs.

Model	$\mathbb{E}_{\mathbf{x}}\|\nabla f(\mathbf{x})\|_2$	$\mathbb{E}_{\mathbf{x}}\|\nabla^2 f(\mathbf{x})\|_2$	$\mathbb{E}_{\mathbf{x}}\mathcal{C}_f(\mathbf{x})$	Accuracy (%)
Standard	19.66 $\pm$ 0.33	6061.96 $\pm$ 968.05	270.89 $\pm$ 75.04	77.42 $\pm$ 0.11
LCNNs	22.04 $\pm$ 1.41	1143.62 $\pm$ 99.38	69.50 $\pm$ 2.41	77.30 $\pm$ 0.11
GradReg [31]	**8.86** $\pm$ 0.12	776.56 $\pm$ 63.62	89.47 $\pm$ 5.86	77.20 $\pm$ 0.26
LCNNs + GradReg	9.87 $\pm$ 0.27	**154.36** $\pm$ 0.22	**25.30** $\pm$ 0.09	77.29 $\pm$ 0.07
CURE [6]	**8.86** $\pm$ 0.01	979.45 $\pm$ 14.05	116.31 $\pm$ 4.58	76.48 $\pm$ 0.07
Softplus + Wt. Decay [7]	18.08 $\pm$ 0.05	1052.84 $\pm$ 7.27	70.39 $\pm$ 0.88	77.44 $\pm$ 0.28
Adversarial Training [32]	**7.99** $\pm$ 0.03	501.43 $\pm$ 18.64	63.79 $\pm$ 1.65	76.96 $\pm$ 0.26

5.2 Impact of Curvature on Gradient Robustness

In §4.1, we showed that low-curvature models must have robust gradients. Here we evaluate whether this is true in practice by measuring the relative gradient robustness for the models with various ranges of curvature values and noise levels. In particular, we measure robustness to random noise



(a) Gradient Robustness on CIFAR10

(b) Gradient Robustness on CIFAR100

Figure 2: Plot showing relative gradient robustness $\frac{\|\nabla f(\mathbf{x}+\epsilon)-\nabla f(\mathbf{x})\|_2}{\|\nabla f(\mathbf{x})\|_2}$ as a function of added noise $\|\epsilon\|_2$ on (a) CIFAR10 and (b) CIFAR100 with a ResNet-18 model. We observe that low-curvature models lead to an **order of magnitude** improvement in gradient robustness, and this improvement closely follows the trend predicted by the theoretical upper bound in §4.

at fixed magnitudes ranging logarithmically from 1×10^{-3} to 1×10^{-1}. We plot our results in Figure 2 where we find that our results match theory (i.e, the simplified quadratic approximation in §4.1) quite closely in terms of the overall trends, and that low curvature models have an **order of magnitude** improvement in robustness over standard models. The close correspondence with theory also shows that deep neural networks can be well-approximated locally by a quadratic.

5.3 Impact of Curvature on Adversarial Robustness

Our theory in §4.2 shows that having low curvature is necessary for robustness, along with having small gradient norms. In this section we evaluate this claim empirically, by evaluating adversarial examples via ℓ_2 PGD [12] adversaries with various noise magnitudes. We use the Cleverhans library [33] to implement PGD. Our results are present in Table 2 where we find that LCNN+GradReg models perform on par with adversarial training, without a resulting accuracy loss.

Table 2: Results indicating off-the-shelf model accuracies (%) upon using ℓ_2 PGD adversarial examples across various noise magnitudes. Adversarial training performs the best overall, however sacrifices clean accuracy. LCNN+GradReg models perform similarly but without significant loss of clean accuracy. Results are averaged across two runs.

Model	Acc. (%)	$\|\epsilon\|_2 = 0.05$	$\|\epsilon\|_2 = 0.1$	$\|\epsilon\|_2 = 0.15$	$\|\epsilon\|_2 = 0.2$
Standard	77.42 $\pm$ .10	59.97 $\pm$ .11	37.55 $\pm$ .13	23.41 $\pm$ .08	16.11 $\pm$ .21
LCNN	77.16 $\pm$ .07	61.17 $\pm$ .53	39.72 $\pm$ .17	25.60 $\pm$ .32	17.66 $\pm$ .18
GradReg	77.20 $\pm$ .26	71.90 $\pm$ .11	61.06 $\pm$ .03	49.19 $\pm$ .12	38.09 $\pm$ .47
LCNNs + GradReg	77.29 $\pm$ .26	**72.68** $\pm$ .52	**63.36** $\pm$ .39	**52.96** $\pm$ .76	**42.70** $\pm$ .77
CURE [6]	76.48 $\pm$ .07	71.39 $\pm$ .12	61.28 $\pm$ .32	49.60 $\pm$ .09	39.04 $\pm$ .16
Softplus + Wt. Decay [7]	77.44 $\pm$ .28	60.86 $\pm$ .36	38.04 $\pm$ .43	23.85 $\pm$ .33	16.20 $\pm$ .01
Adversarial Training [32]	76.96 $\pm$ .26	**72.76** $\pm$ .15	**64.70** $\pm$ .20	**54.80** $\pm$ .25	**44.98** $\pm$ .57

5.4 Train-Test Discrepancy in Model Geometry

During our experiments, we observed a consistent phenomenon where the gradient norms and Hessian norms for test data points were much larger than those for the train data points, which hints at a form of overfitting with regards to these quantities. We term this phenomenon as the *train-test discrepancy* in model geometry. Interestingly, we did not observe any such discrepancy using our proposed curvature measure, indicating that our proposed measure may be a more reliable measure of model geometry. We report our results in Table 3, where we measure the relative discrepancy,

finding that the discrepancy for our proposed measure of curvature is multiple orders of magnitude smaller than the corresponding quantity for gradient and Hessian norms. We leave further investigation of this phenomenon – regarding why curvature is stable across train and test – as a topic for future work.

Table 3: Train-test descrepancy in model geometry, where the relative descrepancy $\Delta_{tt} g(\mathcal{X}) = |\frac{g(\mathcal{X}_{test}) - g(\mathcal{X}_{train})}{g(\mathcal{X}_{test})}|$ is shown for three different geometric measures. We observe that (1) there exists a large train-test descrepancy, with the test gradient / hessian norms being $> 10\times$ the corresponding values for the train set. (2) the descrepancy is 2-3 orders of magnitude smaller for our proposed curvature measure, indicating that it may be a stable model property.

Model	$\Delta_{tt}\mathbb{E}_{\mathbf{x}\in\mathcal{X}}\|\nabla f(\mathbf{x})\|_2$	$\Delta_{tt}\mathbb{E}_{\mathbf{x}\in\mathcal{X}}\|\nabla^2 f(\mathbf{x})\|_2$	$\Delta_{tt}\mathbb{E}_{\mathbf{x}\in\mathcal{X}}\mathcal{C}_f(\mathbf{x})$
Standard	11.75	12.28	**0.025**
GradReg	11.33	11.22	**0.017**
LCNN	19.99	11.33	**0.129**
LCNNs + GradReg	21.82	10.43	**0.146**

Summary of Experimental Results Overall, our experiments show that:

(1) LCNNs have lower curvature than standard models as advertised, and combining them with gradient norm regularization further decreases curvature (see Table 1). The latter phenomenon is unexpected, as our curvature measure ignores gradient scaling.

(2) LCNNs combined with gradient norm regularization achieve an order of magnitude improved gradient robustness over standard models (see Figure 2).

(3) LCNNs combined with gradient norm regularization outperform adversarial training in terms of achieving a better predictive accuracy at a lower curvature (see Table 1), and are competitive in terms of adversarial robustness (see Table 2), while being $\sim 1.4\times$ faster.

(4) We observe that there exists a train-test discrepancy for standard geometric quantities like the gradient and Hessian norm, and this discrepancy disappears for our proposed curvature measure (see Table 3).

We also present ablation experiments, additional adversarial attacks, and evaluations on more datasets and architectures in the Appendix.

6 Discussion

In this paper, we presented a modular approach to remove excess curvature in neural network models. Importantly, we found that combining vanilla LCNNs with gradient norm regularization resulted in models with the smallest curvature, the most stable gradients as well as those that are the most adversarially robust.

The current limitations of our approach are that we only consider convolutional and fully connected layers, and not self-attention or recurrent layers. We also do not investigate the learning-theoretic benefits (or harms) of low-curvature models, or characterize how they may improve generalization for small number of training samples, or provide robustness to label noise (which we already observe in Fig. 1b). Investigating these are important topics for future work.

Acknowledgments and Disclosure of Funding

The authors would like to thank the anonymous reviewers for their helpful feedback and all the funding agencies listed below for supporting this work. SS and HL are supported in part by the NSF awards #IIS-2008461 and #IIS-2040989, and research awards from Google, JP Morgan, Amazon, Harvard Data Science Initiative, and D^3 Institute at Harvard. KM and SS (partly) are supported by the Swiss National Science Foundation under grant number FNS-188758 "CORTI". HL would like to thank Sujatha and Mohan Lakkaraju for their continued support and encouragement.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[2] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[3] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[4] Alexander D'Amour, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen, Jonathan Deaton, Jacob Eisenstein, Matthew D Hoffman, et al. Underspecification presents challenges for credibility in modern machine learning. *arXiv preprint arXiv:2011.03395*, 1(3), 2020.

[5] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1019–1028. PMLR, 06–11 Aug 2017. URL `https://proceedings.mlr.press/v70/dinh17b.html`.

[6] Seyed Mohsen Moosavi Dezfooli, Alhussein Fawzi, Jonathan Uesato, and Pascal Frossard. Robustness via curvature regularization, and vice versa. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9070–9078, Los Alamitos, CA, USA, jun 2019. IEEE Computer Society. URL `https://doi.ieeecomputersociety.org/10.1109/CVPR.2019.00929`.

[7] Ann-Kathrin Dombrowski, Christopher J. Anders, Klaus-Robert Müller, and Pan Kessel. Towards robust explanations for deep neural networks. *Pattern Recognition*, 121:108194, 2022. ISSN 0031-3203. URL `https://doi.org/10.1016/j.patcog.2021.108194`.

[8] Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. *Advances in neural information processing systems*, 30, 2017.

[9] Chongli Qin, James Martens, Sven Gowal, Dilip Krishnan, Krishnamurthy Dvijotham, Alhussein Fawzi, Soham De, Robert Stanforth, and Pushmeet Kohli. Adversarial robustness through local linearization. *Advances in Neural Information Processing Systems*, 32, 2019.

[10] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014. URL `http://arxiv.org/abs/1312.6199`.

[11] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv e-prints*, Dec 2014. URL `http://arxiv.org/abs/1412.6572`.

[12] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJzIBfZAb`.

[13] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pages 1310–1320. PMLR, 2019.

[14] Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile. *Proceedings of the AAAI conference on artificial intelligence*, 33(01):3681–3688, 2019. URL `https://doi.org/10.1609/aaai.v33i01.33013681`.

[15] Xinyang Zhang, Ningfei Wang, Hua Shen, Shouling Ji, Xiapu Luo, and Ting Wang. Interpretable deep learning under fire. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020.

[16] Suraj Srinivas and Francois Fleuret. Rethinking the role of gradient-based attribution methods for model interpretability. In *International Conference on Learning Representations*, 2020.

[17] Andrew Slavin Ros and Finale Doshi-Velez. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018. ISBN 978-1-57735-800-8.

[18] Ann-Kathrin Dombrowski, Maximillian Alber, Christopher Anders, Marcel Ackermann, Klaus-Robert Müller, and Pan Kessel. Explanations can be manipulated and geometry is to blame. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/bb836c01cdc9120a9c984c525e4b1a4a-Paper.pdf`.

[19] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval Networks: Improving Robustness to Adversarial Examples. *arXiv e-prints*, art. arXiv:1704.08847, April 2017.

[20] Asher Trockman and J. Zico Kolter. Orthogonalizing convolutional layers with the cayley transform. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=Pbj8H_jEHYv`.

[21] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. *arXiv e-prints*, Feb 2018. URL `http://arxiv.org/abs/1802.05957`.

[22] Ernest Ryu, Jialin Liu, Sicheng Wang, Xiaohan Chen, Zhangyang Wang, and Wotao Yin. Plug-and-play methods provably converge with properly trained denoisers. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5546–5557. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/ryu19a.html`.

[23] John M Lee. *Riemannian manifolds: an introduction to curvature*, volume 176. Springer Science & Business Media, 2006.

[24] Miaoyan Wang, Khanh Dao Duc, Jonathan Fischer, and Yun S. Song. Operator norm inequalities between tensor unfoldings on the partition lattice. *Linear Algebra and its Applications*, 520:44–66, 2017. ISSN 0024-3795. URL `https://doi.org/10.1016/j.laa.2017.01.017`.

[25] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, and David Duvenaud. Scalable reversible generative models with free-form continuous dynamics. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJxgknCcK7`.

[26] Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J Cree. Regularisation of neural networks by enforcing lipschitz continuity. *Machine Learning*, 110(2):393–416, 2021.

[27] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International conference on machine learning*, pages 274–283. PMLR, 2018.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. URL `https://doi.org/10.1109/CVPR.2016.90`.

[29] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[30] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS 2017 Workshop Autodiff Program*, 2017.

[31] H. Drucker and Y. Le Cun. Improving generalization performance using double backpropagation. *IEEE Transactions on Neural Networks*, 3(6):991–997, 1992. doi: 10.1109/72.165600.

[32] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. *arXiv e-prints*, June 2017. URL http://arxiv.org/abs/1706.06083.

[33] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. Technical report on the CleverHans v2.1.0 adversarial examples library. *arXiv preprint arXiv:1610.00768*, 2018. URL https://arxiv.org/abs/1610.00768.

[34] Sahil Singla and Soheil Feizi. Second-order provable defenses against adversarial attacks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8981–8991. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/singla20a.html.

[35] Shmuel Friedland and Lek-Heng Lim. Nuclear norm of higher-order tensors. *Mathematics of Computation*, 87:1255–1281, 2018. doi: https://doi.org/10.1090/mcom/3239.

[36] Sahil Singla and Soheil Feizi. Fantastic four: Differentiable bounds on singular values of convolution layers. *CoRR*, abs/1911.10258, 2019. URL http://arxiv.org/abs/1911.10258.

[37] Kevin Scaman and Aladin Virmaux. Lipschitz regularity of deep neural networks: Analysis and efficient estimation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 3839–3848, Red Hook, NY, USA, 2018. Curran Associates Inc. URL https://dl.acm.org/doi/10.5555/3327144.3327299.

[38] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. *CoRR*, abs/1702.01135, 2017. URL http://arxiv.org/abs/1702.01135.

[39] Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark, 2021.

[40] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. *CoRR*, abs/2003.01690, 2020. URL https://arxiv.org/abs/2003.01690.

[41] Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search. *CoRR*, abs/1912.00049, 2019. URL http://arxiv.org/abs/1912.00049.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

(b) Did you describe the limitations of your work? [Yes]

(c) Did you discuss any potential negative societal impacts of your work? [N/A] We present foundational work on the geometry of neural network models and as such we do not foresee any negative societal impacts beyond those that already exist with the usage of neural network models.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

   (a) Did you state the full set of assumptions of all theoretical results? [Yes]

   (b) Did you include complete proofs of all theoretical results? [Yes] proofs are provided in the appendix

3. If you ran experiments...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]

   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] some training details are mentioned in the appendix

   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

   (a) If your work uses existing assets, did you cite the creators? [Yes]

   (b) Did you mention the license of the assets? [Yes]

   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...
   We did not use crowdsourcing or conducted research with human subjects.

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Proof of Theorem 1

In this section, we provide the proof of the main theorem in the paper, which decomposes overall curvature into curvatures and slopes of constituent layers. We state Theorem 1 below for reference.

Theorem 2. *Given a function $f = f_L \circ f_{L-1} \circ \ldots \circ f_1$ with $f_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$, the curvature $\mathcal{C}_f$ can be bounded as follows*

$$\mathcal{C}_f(\mathbf{x}) \leq \sum_{i=1}^{L} n_i \times \mathcal{C}_{f_i}(\mathbf{x}) \prod_{j=1}^{i} \|\nabla_{f_{j-1}} f_j(\mathbf{x})\|_2 \leq \sum_{i=1}^{L} n_i \times \max_{\mathbf{x}'} \mathcal{C}_{f_i}(\mathbf{x}') \prod_{j=1}^{i} \max_{\mathbf{x}'} \|\nabla_{f_{j-1}} f_j(\mathbf{x}')\|_2. \tag{6}$$

This statement is slightly different than the one given in the paper, differing by a term of the width of each nonlinear layer. Since we do not use or care about the units of curvature, only its minimization, and have elected to equally-weight each term of the sum, this is an inconsequential discrepancy.

A similar bound is constructed recursively in [34]. [7] gives a similar formula, albeit for the Frobenius norm. The Frobenius norm is both simpler, because the sum of squared entries is independent of the layout of the data, and also weaker, since it cannot deliver a bound which holds uniformly in the data. To our knowledge Equation 6 is the first explicit, easily-interpreted formula of its type.

We start with some preliminaries, with the actual proof being in subsection A.4.

A.1 Derivatives of compositional functions

For a function $f : \mathbb{R}^d \to \mathbb{R}^r$, let $\nabla f : \mathbb{R}^d \to \mathbb{R}^{d \times r}$ denote its gradient, and $\nabla^2 f : \mathbb{R}^d \to \mathbb{R}^{d \times r \times d}$ denote its Hessian. We drop the argument to functions when possible, and all norms will be spectral norms.

Given L functions $f_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}, i = 1 \ldots, L$ let $f_{k,k+j} = f_{k+j} \circ f_{k+j-1} \circ \ldots \circ f_k : \mathbb{R}^{n_{k-1}} \to \mathbb{R}^{n_{k+j}}$ for $1 \leq k \leq k+j \leq L$. This function composition will be our model of a deep neural network where f_ℓ represents the action of the ℓth layer.

If each f_i is continuously differentiable, we have this formula for the gradient of $f_{k,k+j}$

$$\nabla f_{k,k+j} = \prod_{i=1}^{j} \nabla f_{k+j-i+1} \in \mathbb{R}^{n_{k+j} \times n_k}. \tag{7}$$

where we adopt the convention that $f_{j,j}(x) = x$ in order to make the boundary conditions correct. The product begins at the end, with ∇f_{k+j} and progresses forward through the indices – a straightforward consequence of the chain rule of differentiation. Supposing moreover that each f_i is twice-differentiable, we have this formula for the second derivative of $f_{1,k}$:

$$\begin{gathered} \nabla^2 f_{1,k} = \sum_{i=1}^{k} \nabla^2 f_{1,i} - \nabla^2 f_{1,i-1} \\ \text{where } (\nabla^2 f_{1,i} - \nabla^2 f_{1,i-1}) = (\nabla^2 f_i) \left(\nabla f_{1,i-1}, \nabla f_{i+1,k}^\top, \nabla f_{1,i-1} \right) \in \mathbb{R}^{n_0 \times n_k \times n_0} \end{gathered} \tag{8}$$

where we have used the *covariant multilinear matrix multiplication* notation: $\nabla^2 f_i$ is an order-three tensor $\in \mathbb{R}^{n_i \times n_i \times n_i}$, with the first and third modes multiplied by $\nabla f_{1,i-1} \in \mathbb{R}^{n_i \times n_0}$ and the second mode multiplied by $\nabla f_{i+1,k}^\top \in \mathbb{R}^{n_i \times n_k}$.

A.2 Tensor calculus

In this section, we present a simplifed version of the notation from [24]. A k-linear map is a function of k (possibly multidimensional) variables such that if any $k-1$ variables are held constant, the map is linear in the remaining variable. A k-linear function is can be represented by an order-k tensor $\mathcal{A}$ given elementwise by $\mathcal{A} = [\![a_{j_1 \ldots j_k}]\!] \in \mathbb{R}^{d_1 \times \ldots \times d_k}$.

The covariant multilinear matrix multiplication of a tensor with matrices (order 2 tensors) $M_1 = (m^{(1)}_{i_1 j_1}) \in \mathbb{R}^{d_1 \times s_1}, \ldots, M_k = (m^{(k)}_{i_k j_k}) \in \mathbb{R}^{d_k \times s_k}$ is

$$\mathcal{A}(M_1, \ldots, M_k) = \left[\!\!\left[\sum_{i_1=1}^{d_1} \cdots \sum_{i_k}^{d_k} a_{i_1 \ldots i_k} m^{(1)}_{i_1 j_1} \ldots m^{(k)}_{i_k j_k} \right]\!\!\right] \in \mathbb{R}^{s_1 \times \ldots \times s_k}.$$

This operation can be implemented via iterated einsums as:

```
def covariant_multilinear_matmul(a: torch.Tensor,
                                 mlist: List[torch.Tensor]) -> torch.Tensor:
    order = a.ndim
    base_indices = string.ascii_letters
    indices = base_indices[:order]
    next_index = base_indices[order]

    val = a
    for idx in range(order):
        resp_str = indices[:idx] + next_index + indices[idx+1:]
        einsum_str = indices + f",{indices[idx]}{next_index}->{resp_str}"
        val = torch.einsum(einsum_str, val, mlist[idx])
    return val
```

For example, covariant multilinear matrix multiplication of an order two tensor is pre- and post-multiplication by its arguments: $M_1^\top \mathcal{A} M_2 = \mathcal{A}(M_1, M_2)$. The generalization of the matrix spectral norm is $||\mathcal{A}||_2 = \sup\{\mathcal{A}(x_1, \ldots, x_k) : ||x_i|| = 1, x_i \in \mathbb{R}^{d_i}, i = 1, 2, \ldots, k\}$. The computation of order-k operator norms is hard in theory, and also in practice (cf. [35]). In order to address this difficulty, we introduce an instance of the *unfold* operator.

For $\mathcal{A} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$, $\text{unfold}_{\{\{1,2\},\{3\}\}}(\mathcal{A}) \in \mathbb{R}^{d_1 d_2 \times d_3}$ is the matrix with the jth column being the flattened jth (in the final index) $d_2 \times d_3$ matrix.[1] Unfolding is useful because it it allows us to bound an order-3 operator norm in terms of order-2 operator norms – Wang et al. [24] shows that $||\mathcal{A}|| \leq ||\text{unfold}_{\{\{1,2\},\{3\}\}}(\mathcal{A})||$. The upper bound – the operator norm of a *matrix* – can computed with standard largest singular-value routines. A similar unfolding-based bound was used in the deep-learning context by [36] to give improved estimates on the spectral norm of convolution operators.

To facilitate the analysis of unfolded tensors, we coin operations that *put to* the diagonal of tensors:

- $\text{pdiag2} : \mathbb{R}^d \mapsto \mathbb{R}^{d \times d}$ defined by $\text{pdiag2}(x)_{ij} = \begin{cases} x_j & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$
- $\text{pdiag3} : \mathbb{R}^d \mapsto \mathbb{R}^{d \times d \times d}$ defined by $\text{pdiag3}(x)_{ijk} = \begin{cases} x_j & \text{if } i = j = k \\ 0 & \text{otherwise.} \end{cases}$

Further, let $1_n \in \mathbb{R}^n$ be a vector of ones, $I_n = \text{pdiag2}(1_n) \in \mathbb{R}^{n \times n}$ be the n-dimensional identity matrix, and $\mathcal{I}_n = \text{pdiag3}(1_n) \in \mathbb{R}^{n \times n \times n}$. For two vectors $a \in \mathbb{R}^n, b \in \mathbb{R}^n$, let ab denote the elementwise product. $\otimes$ denotes the well-understood Kronecker product, so that, for example, $1_n^\top \otimes I_m$ is an $m \times nm$ matrix consisting of n copies of the $m \times m$ identity matrix stacked side by side. Where it is redundant, we drop the subscripts indicating dimension.

We use the following facts about tensors, their unfoldings, and their operator norms, in what follows.

1. $\mathcal{A} = \text{pdiag3}(ab) \implies \mathcal{A}(M_1, M_2, M_3) = \mathcal{I}(\text{pdiag2}(a)M_1, M_2, \text{pdiag2}(b)M_3)$
2. $\text{unfold}_{\{\{1,2\},\{3\}\}}(\mathcal{I}(M_1, M_2, M_3)) = (M_1 \otimes I_{s_2})^\top \text{unfold}_{\{\{1,2\},\{3\}\}}(\mathcal{I}(I_{d_1}, M_2, M_3))$
3. $||(1_{d_1}^\top \otimes I_{s_2})\text{unfold}_{\{\{1,2\},\{3\}\}}(\mathcal{I}(I_{d_1}, M_2, M_3))|| \leq ||M_2^\top M_3|| s_2$

[1] In PyTorch notation, $\text{unfold}_{\{\{1,2\},\{3\}\}}(\texttt{a}) = \texttt{torch.flatten(a, end_dim=1)}$

Taken together Fact #2 and #3 imply that

$$||\text{unfold}_{\{\{1,2\},\{3\}\}}(\mathcal{I}(M_1, M_2, M_3))|| \leq ||M_1|| \times ||M_2^\top M_3|| \times s_2 \tag{9}$$

which is our essential bound for the norm of an order-3 tensor in terms of order-2 tensors.

A.3 Hessian increment bound

Let $\sigma(x) = \exp(x)/(1 + \exp(x)))$ denote the (elementwise) logistic function $\mathbb{R}^d \mapsto \mathbb{R}^d$. The derivatives of $s(x; \beta)$ can be written as

$$\nabla s(x; \beta) = \text{pdiag2}(\sigma(\beta x)) \in \mathbb{R}^{d \times d} \tag{10}$$

$$\nabla^2 s(x; \beta) = \text{pdiag3}(\beta\sigma(\beta x)(1 - \sigma(\beta x))) \in \mathbb{R}^{d \times d \times d}. \tag{11}$$

Let the ith softplus layer have coefficient β_i, then the increment from Equation 8, can be bounded as follows:

$$||(\nabla^2 f_i)\left(\nabla f_{1,i-1}, \nabla f_{i+1,k}^\top, \nabla f_{1,i-1}\right)|| \tag{12}$$

$$=||\beta_i \mathcal{I}_{n_i}\left(\text{pdiag2}(1 - \sigma(\beta_i x))\nabla f_{1,i-1}, \nabla f_{i+1,k}^\top, \text{pdiag2}(\sigma(\beta_i x))\nabla f_{1,i-1}\right)|| \tag{13}$$

$$\leq||\beta_i \text{unfold}_{\{\{1,2\},\{3\}\}}(\mathcal{I}_{n_i}(\text{pdiag2}(1 - \sigma(\beta_i x))\nabla f_{1,i-1}, \nabla f_{i+1,k}^\top, \text{pdiag2}(\sigma(\beta_i x))\nabla f_{1,i-1})|| \tag{14}$$

$$\leq||\beta_i(\text{pdiag2}(1 - \sigma(\beta_i x))\nabla f_{1,i-1}|| \times ||\nabla f_{i+1,k}^\top \text{pdiag2}(\sigma(\beta_i x))\nabla f_{1,i-1}|| \times n_i \tag{15}$$

$$\leq n_i \times ||\beta_i(\text{pdiag2}(1 - \sigma(\beta_i x))|| \times ||\nabla f_{1,i-1}|| \times ||\nabla f_{i+1,k}^\top \nabla f_i \nabla f_{1,i-1}|| \tag{16}$$

$$=n_i \times \mathcal{C}_{f_i} \times ||\nabla f_{1,i-1}|| \times ||\nabla f||. \tag{17}$$

Equation 13 follows by Fact #1 above, along with Equation 11. Equation 14 is the standard unfolding bound by [24]. Equation 15 is our main bound on order-3 tensors in terms of order-2 matrices, Equation 9. Equation 16 follows from the Cauchy-Schwartz inequality. The replacement in the last term of the product is Equation 10. Equation 17 rewrites Equation 16 using Equation 7: $f_{i+1,k}^\top \nabla f_i \nabla f_{1,i-1} = \nabla f_{1,k}$ and the definition of the curvature of f_i.

A.4 Putting it together

Ignoring the ϵ term,

$$\mathcal{C}_f(\mathbf{x}) = \frac{\|\nabla^2 f(\mathbf{x})\|}{\|\nabla f(\mathbf{x})\|} \tag{18}$$

$$= \frac{1}{\|\nabla f(\mathbf{x})\|} \left|\left|\sum_{i=1}^{L} \nabla^2 f_{1,i}(\mathbf{x}) - \nabla^2 f_{1,i-1}(\mathbf{x})\right|\right| \tag{19}$$

$$\leq \frac{1}{\|\nabla f(\mathbf{x})\|} \sum_{i=1}^{L} ||\nabla^2 f_{1,i}(\mathbf{x}) - \nabla^2 f_{1,i-1}(\mathbf{x})|| \tag{20}$$

$$\leq \sum_{i=1}^{L} n_i \times \mathcal{C}_{f_i}(\mathbf{x}) \times ||\nabla f_{1,i-1}(\mathbf{x})|| \tag{21}$$

$$\leq \sum_{i=1}^{L} n_i \times \mathcal{C}_{f_i}(\mathbf{x}) \times \prod_{i=1}^{j} ||\nabla f_i(\mathbf{x})|| \tag{22}$$

$$\leq \sum_{i=1}^{L} n_i \times \max_{\mathbf{x}} \mathcal{C}_{f_i}(\mathbf{x}) \times \prod_{i=1}^{j} \max_{\mathbf{x}} ||\nabla f_i(\mathbf{x})|| \tag{23}$$

Equation 19 substitutes Equation 8. Equation 20 is the triangle inequality. Equation 21 is Equation 17, along with cancelling the term of $||\nabla f||$ top and bottom. Equation 22 are Equation 23 are

obvious and a standard simplification in the literature on controlling the Lipschitz constant of neural networks. Because exactly computing the smallest Lipschitz constant of a general neural network is NP-complete, a widely-used baseline measure of Lipschitz-smoothness is rather the product of the Lipschitz constants of smaller components of the network, such as single layers ([37]). □

B Loss curvature vs logit curvature

We have thusfar discussed how to assure a bound on curvature independent of the input x. However, a truly data-independent bound must also not depend on the class label y. In this short section we discuss the main consideration that accompanies this: the differences between loss and (pre-softmax) logit curvature. Let the function including the loss for a class c be denoted by f^c_{loss}, and the logit corresponding to a class $c \in [1, C]$ be f^c_{logit}, and let $f^{1,C}_{\text{logit}}$ denote the vector valued function corresponding to all the logits. Then we have $f^c_{\text{loss}} = \text{lsm}^c \circ f^{1,C}_{\text{logit}}$, where $\text{lsm}^c(\mathbf{x}) = -\log \frac{\exp(\mathbf{x}_c)}{\sum_{i=1}^C \exp(\mathbf{x}_i)} = -\mathbf{x}_c + \log \sum_{i=1}^C \exp(\mathbf{x}_i)$ is the negative log softmax function.

The derivatives of this function are:

$$\frac{d\text{lsm}^c(\mathbf{x})}{d\mathbf{x}_c} = \frac{\exp(\mathbf{x}_c)}{\sum_{i=1}^C \exp(\mathbf{x}_i)} - 1$$
$$\frac{d^2\text{lsm}^c(\mathbf{x})}{d\mathbf{x}_c^2} = \frac{\exp(\mathbf{x}_c)}{\sum_{i=1}^C \exp(\mathbf{x}_i)} \left(1 - \frac{\exp(\mathbf{x}_c)}{\sum_{i=1}^C \exp(\mathbf{x}_i)}\right)$$

whose norm is upper bounded by 1 and 0.25 respectively. This implies that bounding the logit curvature also ensures that the loss curvature is bounded, as the gradients and Hessians of the negative log softmax layer do not explode. In other words, $\mathcal{C}_{f_{\text{loss}}} < \mathcal{C}_{f_{\text{logit}}}$.

However penalizing an upper bound may not be the most efficient way to penalize the loss Hessian. Consider the following decomposition of the loss Hessian, which can be written as $\nabla^2 f(\mathbf{x}) \sim \nabla f_{\text{logit}}(\mathbf{x}) \nabla^2_{f_l} \text{LSE}(\mathbf{x}) \nabla f_{\text{logit}}(\mathbf{x})^\top + \nabla_{f_l} \text{LSE}(\mathbf{x}) \nabla^2 f_{\text{logit}}(\mathbf{x})$, where $\text{LSE}(\mathbf{x})$ is the Log-SumExp function. Thus a more efficient way to penalize the loss Hessians is to both use our LCNN penalties, as well as penalize the gradient norm, which we also find to be true in our experiments.

C γ-Lipschitz Batchnorm

We present here a Pytorch-style pseudo-code for the γ-Lipschitz Batchnorm for clarity.

```
self.bn = nn.BatchNorm2d(in_channels, affine=False)
gamma = nn.Parameter(torch.tensor(10.))
...
# perform spectral normalization for BN in closed form
bn_spectral_norm = torch.max(1 / (self.bn.running_var + 1e-5).sqrt())
one_lipschitz_bn = self.bn(x) / bn_spectral_norm

# multiply normalized BN with a learnable scale
return one_lipschitz_bn * torch.minimum(bn_spectral_norm, 1. + gamma)
```

D Proofs of LCNN Properties

Here we present proofs for the properties linking LCNNs to gradient smoothness and adversarial robustness. To this end, we first prove a lemma that is used in both these results, which states that the gradient norm can rise exponentially in the worst case. In the text below, $B_\epsilon(\mathbf{x})$ denotes the ϵ-ball around $\mathbf{x}$.

Lemma 2. *The ratio of gradient norms at nearby points rises exponentially with distance between those points. For a function f satisfying $\max_{\mathbf{x}' \in B_\epsilon(\mathbf{x})} \mathcal{C}_f(\mathbf{x}') = \delta_\mathcal{C}$, points $\mathbf{x}$ and $\mathbf{x} + \epsilon$ that are $\|\epsilon\|_2 = r$ away, we have*

$$\frac{\|\nabla f(\mathbf{x}+\epsilon)\|_2}{\|\nabla f(\mathbf{x})\|_2} \leq \exp(r\delta_\mathcal{C})$$

Proof. Let $g(\mathbf{x}) = \log \|\nabla f(\mathbf{x})\|_2^2$. Applying a first order Taylor series with Lagrange remainder / mean-value theorem, we have

$$\begin{aligned}
g(\mathbf{x}+\epsilon) - g(\mathbf{x}) &= \nabla g(\mathbf{x}+\xi)^\top \epsilon && \text{(for some } \xi \in B_\epsilon(\mathbf{x})\text{)} \\
\log \frac{\|\nabla f(\mathbf{x}+\epsilon)\|_2^2}{\|\nabla f(\mathbf{x})\|_2^2} &= 2\frac{\nabla f(\mathbf{x}+\xi)\nabla^2 f(\mathbf{x}+\xi)^\top \epsilon}{\|\nabla f(\mathbf{x}+\xi)\|_2^2} \\
&\leq 2\frac{\|\nabla f(\mathbf{x}+\xi)\|_2 \|\nabla^2 f(\mathbf{x}+\xi)\|_2 \|\epsilon\|_2}{\|\nabla f(\mathbf{x}+\xi)\|_2^2} && \text{(Cauchy-Schwartz inequality)} \\
&\leq 2r\delta_\mathcal{C}
\end{aligned}$$

Note that to apply the Cauchy-Schwartz inequality, we first upper bound the term on the right by its 2-norm, which for a scalar is simply its absolute value. Taking exponent on both sides of the final expression and taking square root, we have the intended result.

□

D.1 LCNNs have Robust Gradients

Proposition 3. *Let $\max_{\mathbf{x}' \in B_\epsilon(\mathbf{x})} \mathcal{C}_f(\mathbf{x}') \leq \delta_\mathcal{C}$, then the relative distance between gradients at $\mathbf{x}$ and $\mathbf{x} + \epsilon$ is*

$$\frac{\|\nabla f(\mathbf{x}+\epsilon) - \nabla f(\mathbf{x})\|_2^2}{\|\nabla f(\mathbf{x})\|_2^2} \leq r\delta_\mathcal{C} \exp(r\delta_\mathcal{C}) \sim r\mathcal{C}_f(\mathbf{x}) \quad \textit{(Quadratic Approximation)}$$

Proof. We begin the proof by invoking the Taylor series approximation of $\nabla f(\mathbf{x}+\epsilon)$ at $\mathbf{x}$, and using the explicit Lagrange form of the Taylor error. This is equivalent to using a form of the multivariate mean-value theorem. Let $\xi \in B_\epsilon(\mathbf{x})$, then there exists some ξ such that the following holds

$$\begin{aligned}
\nabla f(\mathbf{x}+\epsilon) - \nabla f(\mathbf{x}) &= \nabla^2 f(\mathbf{x}+\xi)^\top \epsilon && \text{(Taylor Series)} \\
\|\nabla f(\mathbf{x}+\epsilon) - \nabla f(\mathbf{x})\|_2 &\leq \|\nabla^2 f(\mathbf{x}+\xi)\|_2 \|\epsilon\|_2 && \text{(Cauchy- Schwartz inequality)} \\
\frac{\|\nabla f(\mathbf{x}+\epsilon) - \nabla f(\mathbf{x})\|_2}{\|\nabla f(\mathbf{x})\|_2} &\leq \frac{\mathcal{C}_f(\mathbf{x}+\xi_0)\|\nabla f(\mathbf{x}+\xi)\|_2}{\|\nabla f(\mathbf{x})\|_2} r && \textit{(Divide by gradnorm)}
\end{aligned}$$

Plugging in the value of $\|\nabla f(\mathbf{x}+\xi_0)\|_2 / \|\nabla f(\mathbf{x})\|_2$ from Lemma 2, and further upper bounding $\mathcal{C}_f(\mathbf{x}+\xi) \leq \delta_\mathcal{C}$ we have the intended result.

To derive the simplified quadratic approximation, replace the first step in the Taylor series with $\nabla f(\mathbf{x}+\epsilon) - \nabla f(\mathbf{x}) = \nabla^2 f(\mathbf{x})^\top \epsilon$, i.e., use $\xi = 0$. □

D.2 Curvature is Necessary for Robustness

Proposition 4. *Let $\max_{\mathbf{x}' \in B_\epsilon(\mathbf{x})} \mathcal{C}_f(\mathbf{x}') \leq \delta_\mathcal{C}$, then for two points $\mathbf{x}$ and $\mathbf{x} + \epsilon$,*

$$\|f(\mathbf{x}+\epsilon) - f(\mathbf{x})\|_2 \leq r\|\nabla f\|_2 \left(1 + \frac{1}{2} r\delta_\mathcal{C} \exp(r\delta_C)\right) \sim r\|\nabla f\|_2 \left(1 + \frac{1}{2} r\mathcal{C}_f(\mathbf{x})\right) \quad \textit{(Quadratic Approximation)}$$

Proof. Let $\xi \in B_\epsilon(\mathbf{x})$, then there exists some ξ such that the following holds

$$f(\mathbf{x}+\epsilon) - f(\mathbf{x}) = \nabla f(\mathbf{x})^\top \epsilon + \frac{1}{2}\epsilon^\top \nabla^2 f(\mathbf{x}+\xi)\epsilon \qquad \text{(Taylor Series)}$$

$$\|f(\mathbf{x}+\epsilon) - f(\mathbf{x})\|_2 \leq \|\nabla f(\mathbf{x})^\top \epsilon\|_2 + \frac{1}{2}\|\epsilon^\top \nabla^2 f(\mathbf{x}+\xi)\epsilon\|_2 \qquad \text{(triangle inequality)}$$

$$\leq \|\nabla f(\mathbf{x})\|_2 \|\epsilon\|_2 + \frac{1}{2}\lambda_{\max}(\mathbf{x}+\xi)\|\epsilon\|_2^2 \qquad \text{(Cauchy-Schwartz inequality)}$$

$$\leq \|\nabla f(\mathbf{x})\|_2 \|\epsilon\|_2 + \frac{1}{2}\mathcal{C}_f(\mathbf{x}+\xi)\|\epsilon\|_2^2 \|\nabla f(\mathbf{x}+\xi)\|_2 \qquad \text{(Defn of } \mathcal{C}_f(\mathbf{x}+\xi)\text{)}$$

Factoring out $\|\nabla f(\mathbf{x})\|_2 \|\epsilon\|_2$ in the RHS, and using Lemma 2, and further upper bounding $\mathcal{C}_f(\mathbf{x}+\xi) \leq \delta_\mathcal{C}$ we have the intended result.

To derive the simplified quadratic approximation, replace the first step in the Taylor series with $f(\mathbf{x}+\epsilon) - f(\mathbf{x}) = \nabla f(\mathbf{x})^\top \epsilon + \frac{1}{2}\epsilon^\top \nabla^2 f(\mathbf{x})\epsilon$, i.e., use $\xi = 0$.

□

E Experimental Settings

In this section we elaborate on the hyper-parameter settings used for our tuning our models. For the standard ResNet-18, we use standard hyper-parameter settings as indicated in the main paper, and we do not modify this for the other variants. For LCNNs, we chose regularizing constants as $\lambda_\beta = 10^{-4}$ and $\lambda_\gamma = 10^{-5}$. For GradReg, we use $\lambda_{\text{grad}} = 10^{-3}$, and for LCNNs + GradReg, we chose $\lambda_\beta = 10^{-4}$, $\lambda_\gamma = 10^{-5}$, $\lambda_{\text{grad}} = 10^{-3}$. We performed a coarse grid search and chose the largest regularizing constants that did not affect predictive performance.

F Additional Experiments

F.1 Ablation Experiments

In this section, we present ablation studies where we train models with each of the proposed modifications separately, i.e., we train a model with only spectral norm for the convolution layers, γ-Lipschitz Batchnorm or centered softplus. Our results in Table 4 show that for the resnet18 architecture considered,

Table 4: Ablation experiments to study effect of individual modifications to LCNN architectures. We find that while either using only centered softplus or γ-BN suffices in practice to minimize curvature, while spectral norm on the convolutional layers (which is the most expensive modification) may not be necessary.

Model	$\mathbb{E}_\mathbf{x}\mathcal{C}_f(\mathbf{x})$	$\mathbb{E}_\mathbf{x}\|\nabla^2 f(\mathbf{x})\|_2$	$\mathbb{E}_\mathbf{x}\|\nabla f(\mathbf{x})\|_2$	**Accuracy (%)**
ConvSpectralNorm only	358.86	8380.55	23.92	77.55
γ-BN only	65.78	1086.95	20.86	77.33
c-Softplus only	57.49	734.05	16.99	77.31
Standard	270.89	6061.96	19.66	77.42
LCNN	69.40	1143.62	22.04	77.30

(1) performing spectral normalization had no effect of curvature as presumably the batchnorm layers are able to compensate for the lost flexibility, and

(2) either penalizing the batchnorm alone or the softplus alone performs almost as well as LCNN, or sometimes even better in terms of curvature reduction. Note that the most expensive computational step is the spectral norm for the convolutional layers, indicating that avoiding this in practice may yield speedups in practice.

While in practice for Resnet-18 only γ-Lipschitz or centered softplus is sufficient for curvature reduction, in theory we must regularize all components to avoid scaling restrictions in one layer

being compensated by other layers, as dictated by the upper bound. In particular, this means that the strategy of penalizing only a subset of layers may not generalize to other architectures.

F.2 Robustness Evaluation on RobustBench / Autoattack

The attack presented in section 5 was relatively "weak" – a network could be truly susceptible to adversarial attack, but by only testing against a weak adversary, we could fail to notice. Short of a comprehensive verification (e.g. [38]), which is known to be computationally intractable at scale, there is no fully satisfactory way to guarantee robustness. However, one common method to develop confidence in a model is to demonstrate robustness in the face of a standard set of nontrivially capable attackers. This is what we do in Table 5, where we use the Robustbench software library [39] to evaluate both a white-box (having access to the internal details of the model), and a black-box (using only function evaluations) attacks.

Table 5: Adversarial accuracy to standard attacks accessed via Robustbench [39]. "APGD-t" refers to the white box targetted auto PGD attack from [40], "Square" refers to the black-box square attack from [41].

Model	**APGD-t Acc. (%)**	**Square Acc. (%)**	**Clean Acc. (%)**
Standard	22.122	52.874	76.721
LCNN	23.709	52.179	76.602
GradReg	50.294	64.678	76.394
LCNN+GradReg	52.477	64.678	76.622
CURE	50.096	63.488	75.928
Softplus+wt decay	23.907	53.766	76.622
Adv Training	55.155	66.861	75.521
CURE + GradReg	60.810	67.357	74.192
LCNN + GradReg + Adv Training	59.222	66.861	75.382

These experimental results show overall that:

(1) LCNN + GradReg is still on par with adversarial training even against stronger attacks such as APGD-t and Square, as they are with PGD.

(2) Combining LCNN + GradReg with adversarial training (in the last row) further improves robustness at the cost of predictive accuracy.

(3) Combining CURE with GradReg (in the penultimate row) improves robustness at the cost of further deteriorating predictive accuracy.

F.3 Additional Evaluations on more Architectures / Dataset Combinations

We present results on the following dataset - architecture pairs:

(1) In Table 6, we present results on SVHN dataset and VGG-11 model

(2) In Table 7, we present results on SVHN dataset and ResNet-18 model

(3) In Table 8, we present results on CIFAR-100 dataset and VGG-11 model

In all cases, we find that our results are on par with our experiment done on the CIFAR-100 and ResNet-18 setup, which confirms that generality of our approach.

Table 6: Model geometry of VGG-11 models trained on the SVHN test dataset.

Model	$\mathbb{E}_{\mathbf{x}}\|\nabla f(\mathbf{x})\|_2$	$\mathbb{E}_{\mathbf{x}}\|\nabla^2 f(\mathbf{x})\|_2$	$\mathbb{E}_{\mathbf{x}}\mathcal{C}_f(\mathbf{x})$	Accuracy (%)
Standard	2.87	158.29	54.24	96.01
LCNNs	4.04	83.34	30.05	95.61
GradReg [31]	1.85	57.52	33.34	96.03
Adversarial Training [32]	1.25	27.64	24.23	96.37

Table 7: Model geometry of Resnet-18 models trained on the SVHN test dataset.

Model	$\mathbb{E}_{\mathbf{x}}\|\nabla f(\mathbf{x})\|_2$	$\mathbb{E}_{\mathbf{x}}\|\nabla^2 f(\mathbf{x})\|_2$	$\mathbb{E}_{\mathbf{x}}\mathcal{C}_f(\mathbf{x})$	Accuracy (%)
Standard	2.64	204.38	78.22	96.41
LCNNs	2.91	77.78	25.36	96.35
GradReg [31]	1.63	68.22	39.55	96.57
Adversarial Training [32]	1.05	22.96	24.48	96.64

Table 8: Model geometry of VGG-11 models trained on the CIFAR-100 test dataset.

Model	$\mathbb{E}_{\mathbf{x}}\|\nabla f(\mathbf{x})\|_2$	$\mathbb{E}_{\mathbf{x}}\|\nabla^2 f(\mathbf{x})\|_2$	$\mathbb{E}_{\mathbf{x}}\mathcal{C}_f(\mathbf{x})$	Accuracy (%)
Standard	17.07	1482.16	85.81	73.33
LCNNs	15.88	282.06	41.14	73.76
GradReg [31]	10.64	534.71	48.26	72.65
Adversarial Training [32]	6.20	166.73	27.37	71.13