Flatten the Curve: Efficiently Training Low-Curvature Neural Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Deep neural networks suffer from adversarial vulnerability and poor interpretability owing to their high degree of non-linearity, which manifests as a large model curvature. Curvature encodes non-linearity typically via Hessian norms. Low-curvature neural network models can help avoid these problems, but existing methods are expensive to train and often sacrifice predictive accuracy. In this work, we demonstrate low-curvature neural networks (LCNNs) that obtain lower curvature than standard models while exhibiting similar predictive performance, and only marginally increased training time. To achieve this, we minimize a data-independent upper bound on the curvature of a neural network, which decomposes overall curvature in terms of curvatures and slopes of its constituent layers. To efficiently minimize this bound, 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 their standard high-curvature counterparts, all without affecting predictive performance. Our approach is easy to use and can be readily incorporated into existing neural network models to remove their excess curvature.

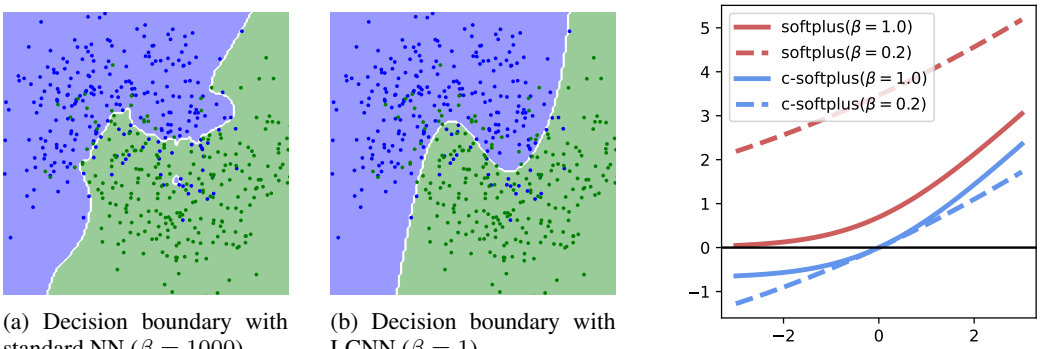
1 Introduction

Deep neural networks are highly non-linear models, and this non-linearity helps performance in difficult tasks such as image classification or machine translation. However, they can have unnecessarily large curvature, which is undesirable because high curvature mappings are susceptible to adversarial vulnerability and unstable gradient maps that hinder interpretability. In this work, we focus on removing excess curvature from neural network models without affecting predictive performance.

Central to this work is a precise notion of curvature, which is a mathematical quantity that encodes the deviation of a curve from 'flatness'. In deep learning, the curvature of a function at a point is often quantified as the norm of the Hessian at that point. For example, Dinh et al. [1] consider the spectral norm of the Hessian while Moosavi-Dezfooli et al. [2] consider the Frobenius norm. In both cases, the corresponding Hessian norm is zero for all inputs if and only if the function is linear. However, a problem with these definitions is that these do not disentangle the model non-linearity from the steepness of model gradients.

To illustrate, let us consider two functions $f(x) = 10^{10}x + \sin(x)$ and $g(x) = 10^{-10}x + \sin(x)$, which have equal Hessians. Intuitively, $f(x) \sim 10^{10}x$ is approximately linear while $g(x) \sim \sin(x)$ is more non-linear. However, if measured by the Hessian, both functions are equally non-linear. This is because while the absolute error with the best fit linear model is the same, $f(x) - 10^{10}x =$

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.



(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.

$g(x) - 10^{-10}x = \sin(x)$, we care more about the *relative* error which appropriately disentangles 'measure of non-linearity' with the magnitude of the linear term. To this end, we propose a modified definition of curvature where the Hessian is scaled by the gradient norm, which we show precisely corresponds to a *relative* local linearity error.

Our overall contributions in this paper are:

1. In §3.1 we propose a novel measure of curvature, dubbed *normalized curvature*, that is invariant to scaling of the gradients.
2. In §3.2 we prove upper bounds on the normalized curvature of neural networks, which decomposes overall curvature in terms of 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.

2 Related Work

Adversarial Robustness of Neural Networks: The well-known phenemenon of adversarial vulnerabilities of neural networks [3, 4] 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 [5] 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. [6] proposed randomized smoothing, which provides a formal guarantee on robustness by generating a smooth classifer from any black-box classifier. Hein and Andriushchenko [7] identified the local Lipschitz constant as critical quantity to prove formal robustness guarantees. Moosavi Dezfooli et al. [8] 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. [10], Zhang et al. [11] 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 [12] 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 [13] showed empirically that robustness can be improved by gradient regularization, however Dombrowski et al. [14] showed that gradient instability is primarily due to large Hessian norms. This suggests that the gradient penalization in Ros and Doshi-Velez [13] performed unintentional Hessian regularization, which is consistent with our experimental results. To alleviate this, Dombrowski et al. [15] proposed to train low curvature models via softplus activations and weight decay, broadly similar to our strategy. However, while Dombrowski et al. [15] focused on penalizing the Frobenius norm of the Hessian, we penalize the the 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. [16] 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 [17] use a reparameterization of the weight matrix, called the Cayley Transform, that is orthogonal by construction. Miyato et al. [18], Ryu et al. [19] 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 Measures

We begin our analysis by discussing a desirable definition of curvature $\mathcal{C}_f(\mathbf{x})$. 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 [8, 14, 1] has been via Hessian norms. An undesirable property of this measure is that it is sensitive to gradient scaling.

For example, if two functions f, g are scaled ($f = k \times g$), or rotated versions of each other ($\nabla_{\mathbf{x}} f = k \times \nabla_{\mathbf{x}} g$), then we would like them to have similar curvatures in the interest of disentangling curvature from scaling. It is easy to see that Hessian norms $\|\nabla^2_{\mathbf{x}} f(\mathbf{x})\|_2$ do not have this property. We would like to be able to recover low-curvature models with steep gradients, precisely to be able to disentangle their properties from low-curvature models with small gradients, which Hessian norms do not allow us to do.

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

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

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.

3.2 A Data-Free Upper Bound on Curvature

Directly penalizing the curvature is computationally expensive as it requires backpropagating from 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. [8] reduce the complexity of this operation by computing a finite-difference approximation to the Hessian from gradients, but even this double-backpropagation is highly demanding.

To address this issue, 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. [20] 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 result is stated below.

Theorem 1. *Given a function $f = f_L \circ f_{L-1} \circ \ldots \circ f_1$, the curvature $\mathcal{C}_f$ can be written as the sum of curvatures of individual layers $\mathcal{C}_{f_i}(\mathbf{x})$, i.e.,*

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

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 most often consist of two kinds of layers 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 constituent layers.

3.3 Centered-Softplus: Activation Function with Trainable Curvature

Theorem 1 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 [21]. The curvature of the softplus function is

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

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 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 ensures that composing softplus functions exacerbates instability around the origin. This issue is exacerbated 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 $s_0(x; \beta \to \infty) = \text{ReLU}(x)$. This ensures that we are able to learn both well-behaved linear maps, and highly non-linear ReLU-like maps if required. Thus, centered-softplus interpolates between linear and nonlinear functions with increasing β and has a finite curvature everywhere.

We further 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 1 shows that controlling the Lipschitz constants of the constituent linear layers in a model is necessary to control overall model curvature. There are broadly three classes of linear layers we consider: convolutions, fully connected, and batch normalization.

3.4.1 Spectrally Normalized Convolutions and Fully Connected Layers

Miyato et al. [18] control the spectral norm of a fully connected layer by reparameterizing – instead of a weight matrix W, rather $W/||W||_2$ is optimized. As long as the maximum singular value is computed accurately, can be done with a small number of iterations of the power method, this approach is fast and simple. Ryu et al. [19] 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 "real" spectral normalization to distinguish it from the approximation that [18] 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 1.

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 the spectral norm of a batch normalization operation is 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

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

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.

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 1 is problematic due to the product of Lipschitz constants of batch normalization layers, which can quickly grow exponentially due to the product term. To alleviate this, we propose to use a penalization $\mathcal{R}_f$ where γ_i terms are aggregated additively across batch normalization layers in the following manner:

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

An additive aggregation ensures that the penalization is well-behaved during training and does not grow exponentially. Further, penalizing the $\log$ of β, γ ensures a gentler regularization, which we found to be stable to train. Note that this achieves the same objective as the upper bound and ensures that the β and γ terms are both small. 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 Advantages of Low Curvature Models

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

4.1 Low Curvature Models have Stable Gradients

Recent work [10, 11] 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*

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

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 $\mathbf{x}$ – 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 $\|\epsilon\|_2 \mathcal{C}_f(\mathbf{x})$.

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 [7]. 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* [22] , which provides an illusion of robustness by making models 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*

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

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 small gradients at data points, the larger 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 ([23]) using the CIFAR10 and CIFAR100 datasets ([24]), and using the Pytorch [25] framework. Our methods entailed fairly modest computation – our most involved computations can be completed in under three days on an NVIDIA GeForce GTX 1080 Ti, and all experimental results could be computed in less than 60 NVIDIA GeForce GTX 1080 Ti-days. We used a mixture of GPUs – primarily NVIDIA GeForce GTX 1080 Tis – on an internal compute cluster.

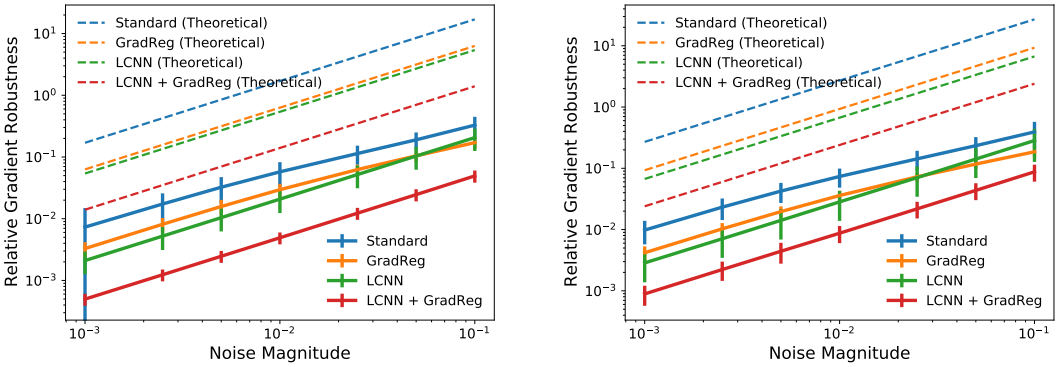
Baselines and Our Methods Our baseline model for comparison involves using a ResNet-18 model with softplus activation with a high $\beta = 10^3$ to mimic ReLU, and yet have well-defined curvatures. Another baseline is gradient norm regularization (which we henceforth call 'GradReg') [26], where the same 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 [8], softplus with weight decay [15] and adversarial training with ℓ_2 PGD [5] 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 control model curvature as intended. 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 control curvature as intended.

We also observe in Table 1 that GradReg [26] 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 $\nabla^2_{\mathbf{x}} f(\mathbf{x}) \sim \nabla_{\mathbf{x}} f_l(\mathbf{x}) \nabla^2_{f_l} \mathrm{LSE}(\mathbf{x}) \nabla_{\mathbf{x}} f_l(\mathbf{x})^\top + \nabla_{f_l} \mathrm{LSE}(\mathbf{x}) \nabla^2_{\mathbf{x}} f_l(\mathbf{x})$, where $\mathrm{LSE}(\mathbf{x})$ 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



(a) Gradient Robustness on CIFAR10

(b) Gradient Robustness on CIFAR100

Figure 2: Plot showing relative gradient robustness $\frac{\|\nabla_{\mathbf{x}} f(\mathbf{x}+\epsilon) - \nabla_{\mathbf{x}} f(\mathbf{x})\|_2}{\|\nabla_{\mathbf{x}} f(\mathbf{x})\|_2}$ 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.

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.

In addition, the average per-epoch training times on a GTX 1080Ti are: standard models / softplus + weight decay ($\sim$ 100 sec), LCNN ($\sim$ 160 sec), GradReg ($\sim$ 270 sec), LCNN+GradReg ($\sim$ 350 sec), CURE / Adversarial Training ($\sim$ 500 sec). Note that the increase in computation for LCNN is primarily due to the use of spectral normalization layers. 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 [26] ('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	$\mathbb{E}_{\mathbf{x}}\|\nabla_{\mathbf{x}} f(\mathbf{x})\|_2$	$\mathbb{E}_{\mathbf{x}}\|\nabla^2_{\mathbf{x}} f(\mathbf{x})\|_2$	$\mathbb{E}_{\mathbf{x}}\mathcal{C}_f(\mathbf{x})$	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 [26]	**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 [8]	**8.86** $\pm$ 0.01	979.45 $\pm$ 14.05	116.31 $\pm$ 4.58	76.48 $\pm$ 0.07
Softplus + Weight Decay [15]	18.08 $\pm$ 0.05	1052.84 $\pm$ 7.27	70.39 $\pm$ 0.88	77.44 $\pm$ 0.28
Adversarial Training [27]	**7.99** $\pm$ 0.03	501.43 $\pm$ 18.64	63.79 $\pm$ 1.65	76.96 $\pm$ 0.26

5.2 Evaluating the 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 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 ones. The close correspondence with theory also shows that deep neural networks can be well-approximated locally by a quadratic.

5.3 Robustness

Our theory in §4.2 shows that having low curvature is necessary along with having small gradient norms for robustness. In this section we evaluate this claim empirically, by evaluating adversarial examples via ℓ_2 PGD [5] adversaries with various noise magnitudes. We use the Cleverhans library [28] to implement PGD. Our results are present in Table 2 where we find that LCNN+Grad 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 trained 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 [8]	76.48 $\pm$ .07	71.39 $\pm$ .12	61.28 $\pm$ .32	49.60 $\pm$ .09	39.04 $\pm$ .16
Softplus + wt. decay [15]	77.44 $\pm$ .28	60.86 $\pm$ .36	38.04 $\pm$ .43	23.85 $\pm$ .33	16.20 $\pm$ .01
Adversarial Training [27]	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	$\Delta_{tt}\mathbb{E}_{\mathbf{x}\in\mathcal{X}}\|\nabla_{\mathbf{x}} f(\mathbf{x})\|_2$	$\Delta_{tt}\mathbb{E}_{\mathbf{x}\in\mathcal{X}}\|\nabla^2_{\mathbf{x}} f(\mathbf{x})\|_2$	$\Delta_{tt}\mathbb{E}_{\mathbf{x}\in\mathcal{X}}\mathcal{C}_f(\mathbf{x})$
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**

6 Discussion

In this paper, we presented a modular approach to remove excess curvature in neural network models. 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.

References

[1] 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`.

[2] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. *arXiv e-prints*, art. arXiv:1610.08401, October 2016.

[3] 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`.

[4] 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`.

[5] 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`.

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

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

[8] 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`.

[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] 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`.

[11] 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.

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

[13] 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.

[14] 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`.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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] (will be submitted by the supplementary materials deadline)
   (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]