Improved techniques for deterministic l_2 robustness

Sahil Singla

Department of Computer Science
University of Maryland
ssingla@umd.edu

Soheil Feizi

Department of Computer Science
University of Maryland
sfeizi@umd.edu

Abstract

Training convolutional neural networks (CNNs) with a strict 1-Lipschitz constraint under the l_2 norm is useful for adversarial robustness, interpretable gradients and stable training. 1-Lipschitz CNNs are usually designed by enforcing each layer to have an orthogonal Jacobian matrix (for all inputs) to prevent the gradients from vanishing during backpropagation. However, their performance often significantly lags behind that of heuristic methods to enforce Lipschitz constraints where the resulting CNN is not *provably* 1-Lipschitz. In this work, we reduce this gap by introducing (a) a procedure to certify robustness of 1-Lipschitz CNNs by replacing the last linear layer with a 1-hidden layer MLP that significantly improves their performance for both standard and provably robust accuracy, (b) a method to significantly reduce the training time per epoch for Skew Orthogonal Convolution (SOC) layers ($> 30\%$ reduction for deeper networks) and (c) a class of pooling layers using the mathematical property that the l_2 distance of an input to a manifold is 1-Lipschitz. Using these methods, we significantly advance the state-of-the-art for standard and provable robust accuracies on CIFAR-10 (gains of $+1.79\%$ and $+3.82\%$) and similarly on CIFAR-100 (+3.78$\%$ and +4.75$\%$) across all networks. Code is available at `https://github.com/singlasahil14/improved_l2_robustness`.

1 Introduction

The Lipschitz constant [1] of a neural network $f : \mathbb{R}^d \rightarrow \mathbb{R}^c$, denoted by $\mathrm{Lip}(f)$, controls the change in the output divided by change in the input (both changes measured in the l_2 norm). Previous work provides evidence that a small Lipschitz constant is useful for interpretable saliency maps [Tsipras et al., 2018], generalization bounds [Long and Sedghi, 2020], Wasserstein distance estimation [Villani, 2008], adversarial robustness [Szegedy et al., 2014] and preventing gradient explosion during backpropagation [Xiao et al., 2018]. Several prior works [Miyato et al., 2018, Gulrajani et al., 2017] use heuristic methods to enforce Lipschitz constraints to successfully address problems such as stabilizing GAN training. However, these methods do not enforce a guaranteed lipschitz bound and it remains challenging to achieve strong results with provable guarantees.

Using the composition property i.e. $\mathrm{Lip}(g \circ h) \leq \mathrm{Lip}(g)\ \mathrm{Lip}(h)$, we can construct a 1-Lipschitz neural network by constraining each layer to be 1-Lipschitz. However, a key difficulty with this approach is that because a 1-Lipschitz layer can only reduce the gradient norm during backpropagation, for deeper networks, this results in small gradient values for layers closer to the input making training slow and difficult. To address this problem, Anil et al. [2018] introduce Gradient Norm Preserving (GNP) architectures where each layer preserves the gradient norm during backpropagation. For 1-Lipschitz Convolutional Neural Networks (CNNs), this involves using orthogonal convolutions (convolution layers with an orthogonal Jacobian matrix) [Li et al., 2019b, Trockman and Kolter,

[1] In this work, we assume the Lipschitz constant under the l_2 norm.

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

Table 1: CIFAR-10 results using faster gradients, projection pooling, CRC-Lip

SOC	Time/epoch (secs)		Reduction	Standard accuracy		Provable robust accuracy	
layers	Ours	Previous		Ours	Previous	Ours	Previous
6	**25.34**	30.63	**-17.27%**	**79.36%**	76.68%	**67.13%**	60.09%
11	**37.11**	48.44	**-23.39%**	**79.57%**	77.73%	**66.75%**	62.82%
16	**49.27**	66.74	**-26.17%**	**79.44%**	77.78%	**66.99%**	62.75%
21	**61.59**	83.18	**-25.96%**	**79.13%**	77.50%	**66.45%**	63.31%
26	**71.51**	100.70	**-28.99%**	**79.19%**	77.18%	**66.28%**	62.46%
31	**84.00**	119.55	**-29.74%**	**78.64%**	74.43%	**66.05%**	59.65%
36	**95.06**	137.10	**-30.66%**	**78.57%**	72.73%	**65.94%**	57.18%
41	**106.01**	156.25	**-32.16%**	**78.41%**	71.33%	**65.51%**	55.74%

2021, Singla and Feizi, 2021, Yu et al., 2022, Kiani et al., 2022] and using a class of GNP activation functions called $\mathrm{HouseHolder}$ activations [Singla et al., 2022].

Among orthogonal convolutions, SOC [Singla and Feizi, 2021] achieves state-of-the-art results on CIFAR-10, 100 [Singla et al., 2022]. SOC uses the following two mathematical properties to construct an orthogonal convolution: (a) the exponential of a skew symmetric matrix is orthogonal and (b) the matrix exponential can be computed using the exponential series. A drawback of using the exponential series is that it requires us to apply the convolution operation multiple times to achieve a reasonable approximation of an orthogonal matrix. Consequently, during backpropagation, computing the gradient with respect to the weights for a single SOC layer requires us to compute the weight gradient for each of these convolutions resulting in significant training overhead. In this work, we show that because the matrix is skew symmetric, using some approximations, the gradients with respect to weights can be computed in significantly reduced time with no loss in performance. This enables us to reduce the training time per epoch using SOC by $> 30\%$ for networks with large number of SOC layers (Results in Table 1).

Another limitation of 1-Lipschitz CNNs is that their performance is often significantly below compared to that of standard CNNs. Recently, Singla et al. [2022] introduced a procedure for certifying robustness by relaxing the orthogonality requirements of the last linear layer (i.e. the linear layer mapping penultimate neurons to class logits) achieving state of the art results on CIFAR-10, 100 [Krizhevsky, 2009]. Since MLPs are more expressive than linear layers, one would expect improved *standard accuracy* by replacing the last linear layer with them. However, the resulting networks are not 1-Lipschitz and achieving high robustness guarantees (*provable robust accuracy*) is difficult.

To certify robustness for these networks, note that since the mapping from input to penultimate layer is 1-Lipschitz, the robustness certificate for the penultimate output also provides a certificate for the input. Thus, we first replace the linear layer mapping penultimate output to logits with a 1-hidden layer MLP (Multi layer Perceptron) because such MLPs are easier to certify compared to deep MLPs. To certify robustness for the MLP, we use the Curvature-based Robustness Certificate or CRC [Singla and Feizi, 2020]. To train MLP to have high robustness guarantees, we use a variant of adversarial training that only applies adversarial perturbations to the MLP (not the whole network). We call our certification procedure $\mathrm{CRC\text{-}Lip}$. This results in improved results for both the standard and also the provable robust accuracy across all network architectures on CIFAR-10 ($\geq 1.63\%$, $\geq 3.14\%$) and CIFAR-100 ($\geq 2.49\%$, $\geq 2.27\%$ respectively). Results are in Tables 3 and 4.

While several works have attempted to construct novel and more expressive orthogonal convolution layers and GNP activation functions, current state-of-the-art 1-Lipschitz CNNs still use pooling layers by taking the $\max$ of different elements. In this work, we introduce a class of 1-Lipschitz pooling layers called *projection pooling* using the following mathematical property: Given a manifold $\mathcal{M} \subset \mathbb{R}^n$ and input $\mathbf{x} \in \mathbb{R}^n$, the function $d_{\mathcal{M}} : \mathbb{R}^n \to \mathbb{R}$ defined as the l_2 distance of $\mathbf{x}$ to $\mathcal{M}$ is 1-Lipschitz. Thus, to construct a pooling layer, we can first define a *learnable manifold* with parameters Θ (Example in Section 6). During training, for input $\mathbf{x} \in \mathbb{R}^n$, the pooling layer simply outputs $d_{\mathcal{M}}(\mathbf{x}) \in \mathbb{R}$ as the output, resulting in decrease in input dimension by a factor of n. Moreover, since the output $d_{\mathcal{M}}(\mathbf{x})$ is also a function of Θ, Θ can be learned during training. However, solving for the distance $d_{\mathcal{M}}(\mathbf{x})$ can be difficult especially when $\mathcal{M}$ is a high-dimensional manifold.

To address this limitation, (a) we use 2D projection pooling layers that reduce the dimension by factor of 2 and (b) we construct these layers using piecewise linear curves for which the distance can be computed efficiently by computing the minimum distance to all line segments and the connecting points (Example in Appendix Figure 2). If the curve is closed and without self-intersections, we can also define a signed projection pooling for which the signs of the output for points inside and outside the region enclosed by the curve are different ($\mathbf{x}$ and $\mathbf{y}$ in Figure 2). This allows the subsequent layers to distinguish between the inputs inside and outside the region. In this work, we show some preliminary results using a simple 2D projection pooling layer (Section 6). We leave the problem of constructing high performance 2D projection pooling layers open for future research.

In summary, in this paper, we make the following contributions:

- We introduce a method for faster computation of the weight gradient for SOC layers. For deeper networks, we observe reduction in training time per epoch by $> 30\%$ (Table 1).
- We introduce a certification procedure called $\mathrm{CRC\text{-}Lip}$ that replaces the last linear layer with a 1-hidden layer MLP and results in significantly improved standard and provable robust accuracy. For deeper networks (≥ 35 layers), we observe improvements of $\geq 5.84\%, \geq 8.00\%$ in standard and $\geq 8.76\%, \geq 8.65\%$ in provable robust accuracy (l_2 radius $36/255$) on CIFAR-10,100 respectively.
- We introduce a large class of 1-Lipschitz pooling layers called *projection pooling* using the mathematical property that the l_2 distance $d_{\mathcal{M}}(\mathbf{x})$ of an input $\mathbf{x}$ to the manifold $\mathcal{M}$ is 1-Lipschitz.
- On CIFAR-10, across all architectures, we achieve the best standard and provable robust accuracy (at $36/255$) of $79.57, 67.13\%$ respectively (gain of $+1.79\%, +3.82\%$ from prior works). Similarly, on CIFAR-100, we achieve $51.84, 39.27\%$ ($+3.78\%, +4.75\%$ from prior works). These results establish new state-of-the-art results in the standard and provable robust accuracy on both datasets.

2 Related work

Provable defenses against adversarial examples: For a provably robust classifier, we can guarantee that its predictions remain constant within some region around the input. Most of the existing methods for provable robustness either bound the Lipschitz constant or use convex relaxations [Singh et al., 2017, 2018a, 2019a,b,c, Weng et al., 2018, Salman et al., 2019b, Zhang et al., 2019, 2018, Wong et al., 2018, Wong and Kolter, 2018, Singh et al., 2018b, Raghunathan et al., 2018, Dvijotham* et al., 2018, Croce et al., 2019, Gowal et al., 2019, Dvijotham* et al., 2020, Lu and Kumar, 2020, Singla and Feizi, 2020, Bunel et al., 2020, Leino et al., 2021, Leino and Fredrikson, 2021, Zhang et al., 2021, 2022, Wang et al., 2021, Huang et al., 2021, Müller et al., 2021, Singh et al., 2021, Palma et al., 2021]. However, these methods are often not scalable to large neural networks while achieving high performance. In contrast, randomized smoothing [Liu et al., 2018, Cao and Gong, 2017, Lécuyer et al., 2018, Li et al., 2019a, Cohen et al., 2019, Salman et al., 2019a, Levine et al., 2019, Kumar et al., 2020a,b, Salman et al., 2020, 2021] scales to large neural networks but is a *probabilistically certified defense*: certifying robustness with high probability requires generating a large number of noisy samples leading to high inference-time. The defense we propose in this work is deterministic and not comparable to randomized smoothing.

Provably Lipschitz neural networks: The class of Gradient Norm Preserving (GNP) and 1-Lipschitz fully connected neural networks was first introduced by Anil et al. [2018]. To design each layer to be GNP, they orthogonalize weight matrices and use a class of piecewise linear GNP activations called $\mathrm{GroupSort}$. Later, Singla et al. [2022] proved that for any piecewise linear GNP function to be continuous, different Jacobian matrices in a neighborhood must change via householder transformations, implying that $\mathrm{GroupSort}$ is a special case of more general $\mathrm{HouseHolder}$ activations. Several previous works enforce Lipschitz constraints on convolution layers using spectral normalization, clipping or regularization [Cissé et al., 2017, Tsuzuku et al., 2018, Qian and Wegman, 2019, Gouk et al., 2020, Sedghi et al., 2019]. However, these methods either enforce loose lipschitz bounds or do not scale to large networks. To ensure that the Lipschitz constraint on convolutional layers is tight, recent works construct convolution layers with an orthogonal Jacobian [Li et al., 2019b, Trockman and Kolter, 2021, Singla and Feizi, 2021, Yu et al., 2022, Su et al., 2022, Kiani et al., 2022]. These approaches avoid the aforementioned issues and allow training of large, provably 1-Lipschitz CNNs achieving state-of-the-art results for deterministic l_2 robustness.

3 Problem setup and Notation

For a vector $\mathbf{v}$, $\mathbf{v}_j$ denotes its j^{th} element. For a matrix $\mathbf{A}$, $\mathbf{A}_{j,:}$ and $\mathbf{A}_{:,k}$ denote the j^{th} row and k^{th} column respectively. Both $\mathbf{A}_{j,:}$ and $\mathbf{A}_{:,k}$ are assumed to be column vectors (thus $\mathbf{A}_{j,:}$ is the transpose of j^{th} row of $\mathbf{A}$). $\mathbf{A}_{j,k}$ denotes the element in j^{th} row and k^{th} column of $\mathbf{A}$. $\mathbf{A}_{:j,:k}$ denotes the matrix containing the first j rows and k columns of $\mathbf{A}$. We define $\mathbf{A}_{:j} = \mathbf{A}_{:j,:}$ and $\mathbf{A}_{j:} = \mathbf{A}_{j:,:}$. Similar notation applies to higher order tensors. $\mathbf{I}$ denotes the identity matrix, $\mathbb{R}$ to denote the field of real numbers. We construct a 1-Lipschitz neural network, $f : \mathbb{R}^d \to \mathbb{R}^c$ (d is the input dimension, c is the number of classes) by composing 1-Lipschitz convolution layers and GNP activation functions. We often use the abbreviation $f_i - f_j : \mathbb{R}^d \to \mathbb{R}$ to denote the function so that:

$$\left(f_i - f_j\right)(\mathbf{x}) = f_i(\mathbf{x}) - f_j(\mathbf{x}), \qquad \forall\, \mathbf{x} \in \mathbb{R}^d$$

For a matrix $\mathbf{A} \in \mathbb{R}^{q \times r}$ and a tensor $\mathbf{B} \in \mathbb{R}^{p \times q \times r}$, $\overrightarrow{\mathbf{A}}$ denotes the vector constructed by stacking the rows of $\mathbf{A}$ and $\overrightarrow{\mathbf{B}}$ by stacking the vectors $\overrightarrow{\mathbf{B}_{j,:,:}}$, $j \in [p-1]$ so that:

$$\left(\overrightarrow{\mathbf{A}}\right)^T = \left[\mathbf{A}_{0,:}^T\,,\ \mathbf{A}_{1,:}^T\,,\ \ldots\ ,\ \mathbf{A}_{q-1,:}^T\right], \quad \left(\overrightarrow{\mathbf{B}}\right)^T = \left[\left(\overrightarrow{\mathbf{B}_{0,:,:}}\right)^T\,,\ \left(\overrightarrow{\mathbf{B}_{1,:,:}}\right)^T\ ,\ \ldots\ ,\ \left(\overrightarrow{\mathbf{B}_{p-1,:,:}}\right)^T\right]$$

For a 2D convolution filter, $\mathbf{L} \in \mathbb{R}^{p \times q \times r \times s}$ and input $\mathbf{X} \in \mathbb{R}^{q \times n \times n}$, we use $\mathbf{L} \star \mathbf{X} \in \mathbb{R}^{p \times n \times n}$ to denote the convolution of filter $\mathbf{L}$ with $\mathbf{X}$. We use the the notation $\mathbf{L} \star^i \mathbf{X} \triangleq \mathbf{L} \star^{i-1} (\mathbf{L} \star \mathbf{X})$ and $\mathbf{L} \star^0 \mathbf{X} = \mathbf{X}$. Unless specified, we assume zero padding and stride 1 in each direction.

4 Faster gradient computation for Skew Orthogonal Convolutions

Among the existing orthogonal convolution layers in the literature, Skew Orthogonal Convolutions (or SOC) by Singla and Feizi [2021] achieves state-of-the-art results on CIFAR-10,100 [Singla et al., 2022]. SOC first constructs a convolution filter $\mathbf{L}$ whose Jacobian is skew-symmetric. This is followed by a convolution exponential [Hoogeboom et al., 2020] operation. Since the exponential of a skew-symmetric matrix is orthogonal, the Jacobian of the resulting layer is an orthogonal matrix.

However, a drawback of this procedure is that convolution exponential requires multiple convolution operations per SOC layer to achieve a reasonable approximation of the orthogonal Jacobian. Consequently, if we use k convolution operations in the SOC layer during forward pass, we need to compute the gradient with respect to the weights $\mathbf{L}$ (called *convolution weight gradient*) per convolution operation (k times) which can lead to slower training time especially when the number of SOC layers is large. To address this limitation, we show that even if we use k convolutions in the forward pass of an SOC layer, the weight gradient for the layer can be computed using a *single convolution weight gradient* during backpropagation leading to significant reduction in training time.

For simplicity, let us first consider the case of an orthogonal fully connected layer (i.e. not convolution) with the same input and output size (n). Later, we will see that our analysis leads to improvements for orthogonal convolutional layers. Further, assume that the weights i.e. $\mathbf{A} \in \mathbb{R}^{n \times n}$ are skew-symmetric i.e. $\mathbf{A} = -\mathbf{A}^T$ and given the input $\mathbf{x} \in \mathbb{R}^n$, the output $\mathbf{z} \in \mathbb{R}^n$ is computed as follows:

$$\mathbf{z} = \left(\sum_{i=0}^{k} \frac{\mathbf{A}^i}{i!}\right)\mathbf{x} + \mathbf{b}, \qquad \text{where} \quad \mathbf{A} = -\mathbf{A}^T \tag{1}$$

We approximate the exponential series: $\exp(\mathbf{A}) = \sum_{i=0}^{\infty} \mathbf{A}^i / i!$ using a finite number of terms (k).
Forward pass ($\mathbf{z}$): To compute $\mathbf{z}$ during the forward pass, we use the following iterations:

$$\mathbf{u}^{(i)} = \begin{cases} \mathbf{x} & i = k-1 \\ \mathbf{x} + (\mathbf{A}\mathbf{u}^{(i+1)})/(i+1) & i \leq k-2 \end{cases} \tag{2}$$

It can be shown that $\mathbf{z} = \mathbf{u}^{(0)}$ (Appendix C). During backpropagation, given the gradient of loss ℓ w.r.t. layer output $\mathbf{z}$ i.e. $\nabla_{\mathbf{z}}\, \ell$, we want to compute $\nabla_{\mathbf{x}}\, \ell$ (input gradient) and $\nabla_{\mathbf{A}}\, \ell$ (weight gradient).

Input gradient ($\nabla_{\mathbf{x}}\, \ell$): To compute $\nabla_{\mathbf{x}}\, \ell$, observe that $\mathbf{z}$ is a linear function of $\mathbf{x}$. Thus, using the chain rule, skew-symmetricity ($\mathbf{A}^T = -\mathbf{A}$) and the property $\left(\mathbf{A}^i\right)^T = \left(\mathbf{A}^T\right)^i$, we have:

$$\nabla_{\mathbf{x}}\, \ell = \left(\sum_{i=0}^{k} \frac{\mathbf{A}^i}{i!}\right)^T (\nabla_{\mathbf{z}}\, \ell) = \left(\sum_{i=0}^{\infty} \frac{(-\mathbf{A})^i}{i!}\right)(\nabla_{\mathbf{z}}\, \ell)$$

We can again approximate the exponential series using the same number of terms as in the forward pass i.e. k. To compute the finite term approximation, we use the following iterations:

$$\mathbf{v}^{(i)} = \begin{cases} \nabla_{\mathbf{z}}\, \ell & i = k-1 \\ \nabla_{\mathbf{z}}\, \ell - (\mathbf{A}\mathbf{v}^{(i+1)})/(i+1) & i \leq k-2 \end{cases} \tag{3}$$

Similar to forward pass, it can be shown that $\nabla_{\mathbf{x}}\, \ell = \mathbf{v}^{(0)}$ (Appendix C).

Weight gradient ($\nabla_{\mathbf{A}}\, \ell$): We first derive the exact expression for $\nabla_{\mathbf{A}}\, \ell$ in the Theorem below:

Theorem 1 *The gradient of the loss function ℓ w.r.t $\mathbf{A}$ i.e. $\nabla_{\mathbf{A}}\, \ell$ is given by:*

$$\nabla_{\mathbf{A}}\, \ell = -\sum_{i=1}^{k} \left(\left(\mathbf{A}^{i-1}\mathbf{x}\right) \left(\mathbf{v}^{(i)}\right)^T - \mathbf{v}^{(i)} \left(\mathbf{A}^{i-1}\mathbf{x}\right)^T \right) \tag{4}$$

where $\mathbf{v}^{(i)}$ is defined as in equation (3).

Note that the first outer product i.e. $\left(\mathbf{A}^{i-1}\mathbf{x}\right)\left(\mathbf{v}^{(i)}\right)^T$ and the second i.e. $\left(\mathbf{v}^{(i)}\right)\left(\mathbf{A}^{i-1}\mathbf{x}\right)^T$ are transpose of each other implying that each term in the summation is skew-symmetric. Although these outer products can be computed in a straightforward way for orthogonal *fully connected* layers, this is not the case for orthogonal *convolution* layers (SOC in this case). This is because, for a convolution filter $\mathbf{L} \in \mathbb{R}^{p \times q \times r \times s}$, the term analogous to $\mathbf{A}^{i-1}\mathbf{x}$ is a patch of size $q \times r \times s$ and that analogous to $\mathbf{v}^{(i)}$ is another patch of size $p \times 1 \times 1$ resulting in an outer product of the desired size $p \times q \times r \times s$ *per patch*. Thus, for SOC layers, each term inside the summation is computed by summing over the outer products for all such patches. For large input sizes, the number of such patches can often be large, making this computation expensive. To address this limitation, we use the following approximation:

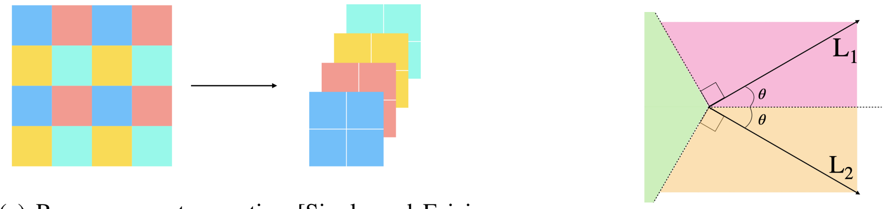
$$\nabla_{\mathbf{A}}\, \ell \approx -\left(\mathbf{u}^{(1)} \left(\mathbf{v}^{(1)}\right)^T - \mathbf{v}^{(1)} \left(\mathbf{u}^{(1)}\right)^T \right) \tag{5}$$

In Appendix B, we show that the above approximation is principled because after subtracting the exact and approximation gradients (equations (4) and (5)) and simplifying, each term in the resulting series is divided by a large value in its denominator (≈ 0). This approximation is useful because in equation (5), the outer product needs to be computed once whereas in equation (4), the outer products need to be computed k times. Also, both $\mathbf{u}^{(1)}$ and $\mathbf{v}^{(1)}$ are computed while computing $\mathbf{u}^{(0)} = \mathbf{z}$ during the forward pass and $\mathbf{v}^{(0)} = \nabla_{\mathbf{x}}\, \mathbf{z}$ during the backward pass using the recurrences in equations (2) and (3). Thus, we can simply store $\mathbf{u}^{(1)}, \mathbf{v}^{(1)}$ during the forward,backward pass respectively so that $\nabla_{\mathbf{A}}\, \ell$ can be computed directly using equation (5). In our experiments, we observe that this leads to significant reduction in training time with almost no drop in performance (Tables 1, 3, 4).

5 Curvature-based Robustness Certificate

A key property of 1-Lipschitz CNNs is that the output of each layer is 1-Lipschitz with respect to the input. Given an input $\mathbf{x} \in \mathbb{R}^d$, consider the penultimate output $g(\mathbf{x}) \in \mathbb{R}^m$ and logits $f(\mathbf{x}) \in \mathbb{R}^c$ for some 1-Lipschitz CNN. Existing robustness certificates [Li et al., 2019b, Singla et al., 2022] rely on the *linearity* of the function from $g(\mathbf{x}) \rightarrow f(\mathbf{x})$. However, since MLPs have higher expressive power than linear functions [Cybenko, 1989], one way to improve performance could be to replace this mapping with MLPs. However, certifying robustness for deep MLPs is difficult.

To address these challenges, we first replace the mapping from $g(\mathbf{x}) \rightarrow f(\mathbf{x})$ with a 1-hidden layer MLP because they are easier to certify compared to deeper networks. Because computing exact certificates for ReLU networks is known to be NP-complete [Sinha et al., 2018], we use the differentiable $\mathrm{Softplus}$ activation [Dugas et al., 2000] to certify robustness. To certify robustness, we use the Curvature-based Robustness Certificate or CRC [Singla and Feizi, 2020] because it provides exact certificates for a significant fraction of inputs for shallow MLPs. We provide a brief review of CRC in Appendix Section D. Let $g : \mathbb{R}^d \rightarrow \mathbb{R}^m$ be a 1-Lipschitz continuous function and $h : \mathbb{R}^m \rightarrow \mathbb{R}^c$ be a 1-hidden layer MLP such that $f = h \circ g$. Further, let l be the predicted class for input $\mathbf{x}$ i.e. $f_l(\mathbf{x}) \geq \max_{i \neq l} f_i(\mathbf{x})$. Since g is 1-Lipschitz, it can be shown that if h is provably robust in an l_2 radius R around input $g(\mathbf{x})$, then f is also provably robust in the l_2 radius R around $\mathbf{x}$. The resulting procedure is called $\mathrm{CRC\text{-}Lip}$ and is given in the following proposition:



(a) Rearrangement operation [Singla and Feizi, 2021]

(b) 2D projection pooling

Figure 1: Illustration of the rearrangement operation (left) and 2D projection pooling (right).

Proposition 1 (CRC-Lip) *For input* $\mathbf{x}$ *such that* $f_l(\mathbf{x}) \geq \max_{i \neq l} f_i(\mathbf{x})$*, let* R *be the robustness certificate for* h *using input* $g(\mathbf{x})$*, then* R *is also the robustness certificate for the function* f*:*

$$\min_{i \neq l} \min_{h_l(\mathbf{y}^*) = h_i(\mathbf{y}^*)} \|\mathbf{y}^* - g(\mathbf{x})\|_2 \geq R \implies \min_{i \neq l} \min_{f_l(\mathbf{x}^*) = f_i(\mathbf{x}^*)} \|\mathbf{x}^* - \mathbf{x}\|_2 \geq R \tag{6}$$

Proof is in Appendix A.2. In our experiments, we find that although replacing the linear layer with an MLP achieves high standard accuracy, directly using the above certificate often leads to very small robustness certificates and thus low certified robust accuracy. To address this problem, we introduce (a) an adversarial training procedure that *only* applies to the input of the MLP $g(\mathbf{x})$ (i.e. not the input of the neural network $\mathbf{x}$) and (b) curvature regularization to reduce the curvature of the MLP. It was shown in Singla and Feizi [2021] that combining adversarial training with curvature regularization leads to significantly improved provable robust accuracy with small reduction in standard accuracy. This results in the following loss function for training:

$$\min_{\Omega,\ \Psi} \quad \mathbb{E}_{(\mathbf{x},l) \sim \mathcal{D}} \left[\left(\max_{\|\mathbf{y}^* - g_{\Psi}(\mathbf{x})\|_2 \leq \rho} \ell\left(h_{\Omega}(\mathbf{y}^*, l)\right) \right) + \gamma\, \mathcal{K}_h \right] \tag{7}$$

In the above equation, Ω denote the parameters of the MLP (i.e. h), Ψ are the parameters of the 1-Lipschitz function g, $\mathcal{K}_h$ is the bound on the curvature of the MLP, ρ denotes the l_2 perturbation radius and γ denotes the curvature regularization coefficient. The curvature bound $\mathcal{K}_h$ is the same as in Singla and Feizi [2020]. Also, since we apply adversarial perturbations in the penultimate layer, we also need to backpropagate through this procedure to enable training of previous layers. To this end, we simply use an identity map (same gradient output as the gradient input) to backpropagate through the adversarial training procedure and find that it works well in practice, achieving significantly better results compared to the state-of-the-art. It is possible that a more principled method of backpropagation may lead to better results and we leave that avenue open for future research.

6 Projection pooling layers

In 1-Lipschitz CNNs, pooling is usually carried out as follows: given input $\mathbf{X} \in \mathbb{R}^{q \times r \times r}$ (r is even), we first use *rearrangement* [Jacobsen et al., 2018] illustrated in Figure 1b to construct $\mathbf{X}' \in \mathbb{R}^{4q \times (r/2) \times (r/2)}$. Next, we apply an orthogonal convolution which gives an output of the same size $\mathbf{Z} \in \mathbb{R}^{4q \times (r/2) \times (r/2)}$ and divide it into two tensors of equal sizes (along the channel dimension) giving $\mathbf{Z}_{:2q} \in \mathbb{R}^{2q \times (r/2) \times (r/2)}$ and $\mathbf{Z}_{2q:} \in \mathbb{R}^{2q \times (r/2) \times (r/2)}$. We then define $\max(\mathbf{Z}_{:2q},\ \mathbf{Z}_{2q:})$ (or either one of $\mathbf{Z}_{:2q},\ \mathbf{Z}_{2q:}$) as the output of the pooling layer.

Although $\max$ is 1-Lipschitz, its expressive power is limited. For example, consider the 1-Lipschitz function $\|x, y\|_2 = \sqrt{x^2 + y^2}$. It is easy to see that if $x = y$, the error between $\|x, y\|_2$ and $\max(x, y)$ can be arbitrarily large for large values of x, y. To address such limitations, we construct expressive pooling layers using the following mathematical property:

Theorem 2 *Given* $\mathbf{x} \in \mathbb{R}^n$ *and manifold* $\mathcal{M} \subset \mathbb{R}^n$*, the distance function (in* l_2 *norm) is 1-Lipschitz:*

$$d_{\mathcal{M}}(\mathbf{x}) = \min_{\mathbf{x}^* \in \mathcal{M}} \|\mathbf{x}^* - \mathbf{x}\|_2 \implies |d_{\mathcal{M}}(\mathbf{x}) - d_{\mathcal{M}}(\mathbf{y})| \leq \|\mathbf{x} - \mathbf{y}\|_2 \tag{8}$$

The above theorem provides a powerful method for constructing a *learnable pooling layer* by selecting a *learnable manifold* $\mathcal{M}_{\Theta} \subset \mathbb{R}^r$ (Θ denotes the set of learnable parameters). To apply the pooling

Output Size	Layer
$32 \times 32 \times 32$	Conv + MaxMin
$64 \times 16 \times 16$	LipBlock-n/5
$128 \times 8 \times 8$	LipBlock-n/5
$256 \times 4 \times 4$	LipBlock-n/5
$512 \times 2 \times 2$	LipBlock-n/5
$1024 \times 1 \times 1$	LipBlock-n/5
# of classes	Linear/MLP-1

(a) LipConvnet-n Architecture

Output Size	Layer	Repeats
$q \times r \times r$	Input	_
$q \times r \times r$	Conv + MaxMin	$\mathrm{n}/5 - 1$
$4q \times (r/2) \times (r/2)$	Rearrange	1
$4q \times (r/2) \times (r/2)$	Conv	1
$2q \times (r/2) \times (r/2)$	Pooling	1

(b) LipBlock-n/5

Table 2: LipConvnet-n and LipBlock-n/5 architectures

operation to an input $\mathbf{x} \in \mathbb{R}^n$, we simply output its l_2 distance to the manifold $\mathcal{M}_\Theta$ denoted by $d_{\mathcal{M}_\Theta}(\mathbf{x})$. Since $d_{\mathcal{M}_\Theta}(\mathbf{x}) \in \mathbb{R}$ while $\mathbf{x} \in \mathbb{R}^n$, this operation reduces the input size by a factor of n.

As an example, let $\mathcal{M}_\mathbf{u} = \{\mathbf{u}\}$. Here $\mathbf{u} \in \mathbb{R}^n$ is the learnable parameter and the distance function $d_{\mathcal{M}_\mathbf{u}}(\mathbf{x}) = \|\mathbf{x} - \mathbf{u}\|_2$. Even in this very simple case, for $\mathbf{u} = \mathbf{0}$, $d_{\mathcal{M}_\mathbf{u}}(\mathbf{x}) = \|\mathbf{x}\|_2$ and for $n = 2$, this function can learn to represent the function $\|x, y\|_2$ discussed earlier exactly. We also prove that a *signed* l_2 distance function can be defined when $\mathcal{M}$ satisfies certain properties:

Corollary 1 *If $\mathcal{M}$ is a connected manifold that divides $\mathbb{R}^n$ into two nonempty connected sets $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \cap \mathcal{B} = \phi$ and every path from $\mathbf{a} \in \mathcal{A}$ and $\mathbf{b} \in \mathcal{B}$ intersects a point on $\mathcal{M}$, then there exists a signed l_2 distance function with different signs in $\mathcal{A}$ and $\mathcal{B}$.*

Proofs of Theorems 2 and Corollary 1 are given in Appendix A.3 and A.4 respectively.

In practice, computing $d_\mathcal{M}$ can be difficult for high-dimensional $\mathcal{M}$. To tackle this issue, we use 2D pooling layers where $\mathcal{M}$ is defined to be a piecewise linear curve in 2D. The l_2 distance can then be computed by finding the minimum distance to all the line segments and connecting points. We emphasize that even in this relatively simple case, $\mathcal{M}$ can have large number of parameters and $d_\mathcal{M}$ can still be efficient to compute because these individual distances can be computed in parallel. We use $\mathcal{M}_\theta$ defined below (illustrated in Figure 1b, lines $L_1,\ L_2$ correspond to $\phi = +\theta, -\theta$):

$$\mathcal{M}_\theta = \{R(\cos\phi,\ \sin\phi): \quad R \geq 0,\ \phi \in \{+\theta, -\theta\}\}$$

In each colored region (Figure 1b), $d_{\mathcal{M}_\theta}$ can be computed using the following:

$$d_{\mathcal{M}_\theta}\left(R(\cos\alpha,\ \sin\alpha)\right) = \begin{cases} R\sin(\alpha - \theta), & 0 \leq \alpha \leq \theta + \pi/2 \\ -R\sin(\alpha + \theta), & 0 < -\alpha \leq \theta + \pi/2 \\ R, & \text{otherwise} \end{cases} \tag{9}$$

To apply projection pooling on $\mathbf{Z}_{:2q},\ \mathbf{Z}_{2q:}$ discussed previously, we output $d_{\mathcal{M}_\theta}\left(\mathbf{Z}_{:2q}, \mathbf{Z}_{2q:}\right)$.

7 Experiments

We perform experiments under the setting of provably robust image classification on CIFAR-10 and CIFAR-100 datasets. We use the LipConvnet-5, 10, 15, …, 40 architectures for comparison. We use SOC as the orthogonal convolution and MaxMin as the activation in all architectures. All experiments were performed using 1 NVIDIA GeForce RTX 2080 Ti GPU. All networks were trained for 200 epochs with initial learning rate of 0.1, dropped by a factor of 0.1 after 100 and 150 epochs. For adversarial training with curvature regularization, we use $\rho = 36/255\ (0.1411),\ \gamma = 0.5$ for CIFAR-10 and $\rho = 0.2$, $\gamma = 0.75$ for CIFAR-100. We find that certifying robustness using CRC-Lip is computationally expensive for CIFAR-100 due to large number of classes. To address this issue, we only use classes with top-10 logits (instead of all 100 classes) for CIFAR-100 (Table 4). Since CRC-Lip requires us to solve a convex optimization, we consider a certificate to be valid if the input is correctly classified and gradient at the optimal solution is $\leq 10^{-6}$ (0 otherwise). All results are

Table 3: Provable robustness results on CIFAR-10 (LipConvnet-$5, 10$ results in Appendix Table 5)

LipConv net-	**Methods**	**Standard Accuracy**	**Provable Robust Accuracy**			**Increase**	
			36/255	72/255	108/255	(standard)	(36/255)
15	Baseline	77.78%	62.75%	46.34%	31.38%	_	_
	+ Fast	77.75%	62.52%	46.23%	31.19%	-0.03%	-0.23%
	+ CRC	**79.44%**	**66.99%**	**52.56%**	**38.30%**	**+1.66%**	**+4.24%**
20	Baseline	77.50%	63.31%	46.42%	31.53%	_	_
	+ Fast	77.13%	62.05%	45.86%	31.13%	-0.37%	-1.26%
	+ CRC	**79.13%**	**66.45%**	**52.45%**	**38.12%**	**+1.63%**	**+3.14%**
25	Baseline	77.18%	62.46%	45.78%	31.16%	_	_
	+ Fast	76.94%	61.91%	45.59%	30.69%	-0.24%	-0.55%
	+ CRC	**79.19%**	**66.28%**	**51.74%**	**37.99%**	**+2.01%**	**+3.82%**
30	Baseline	74.43%	59.65%	43.76%	29.16%	_	_
	+ Fast	74.69%	58.84%	43.33%	28.93%	+0.26%	-0.81%
	+ CRC	**78.64%**	**66.05%**	**51.31%**	**37.30%**	**+4.21%**	**+6.40%**
35	Baseline	72.73%	57.18%	42.08%	28.09%	_	_
	+ Fast	72.91%	57.58%	41.52%	27.37%	+0.18%	+0.40%
	+ CRC	**78.57%**	**65.94%**	**52.04%**	**37.63%**	**+5.84%**	**+8.76%**
40	Baseline	71.33%	55.74%	39.32%	26.06%	_	_
	+ Fast	71.60%	56.15%	39.82%	25.63%	+0.27%	+0.41%
	+ CRC	**78.41%**	**65.51%**	**51.32%**	**37.30%**	**+7.08%**	**+9.77%**

reported using the complete test sets of CIFAR-10 and CIFAR-100. We compare the provable robust accuracy using 3 different l_2 perturbation radii: $36/255,\ 72/255,\ 108/255$.

Table details: For the baseline ("Baseline" in Tables 3, 4), we use the standard $\max$ pooling with the certificate based on LLN [Singla et al., 2022] due to their superior performance over prior works. In Tables 3 and 4, for each architecture, "**+ Fast**" adds faster gradient computation, "**+ CRC**" replaces $\max$ pooling with projection pooling (equation (9)) and replaces the last linear layer with a 1-hidden layer MLP (CRC-Lip certificate) while also using faster gradients. For each architecture, the columns "Increase (Standard)" and "Increase (36/255)" denote the increase in standard and provable robust accuracy relative to "Baseline" standard and provable robust accuracy $(36/255)$. Results where Projection pooling and CRC-Lip are added separately are given in Appendix Tables 5 and 6.

LipConvnet Architecture: We use a 1-Lipschitz CNN architecture called $\mathrm{LipConvnet}$-n where n is a multiple of 5 and $\mathrm{n} + 1$ is the total number of convolution layers. It consists of an initial SOC layer that expands the number of channels from 3 to 32. This is followed by 5 blocks that reduce spatial dimensions (height and width) by half while doubling the number of channels. The architecture is summarized in Tables 2a and 2b. The last layer outputs the class logits and is either a linear layer in which case we use LLN to certify robustness or a single-hidden layer MLP where we use CRC-Lip.

Correcting the certificates: Since SOC is an approximation, the Lipschitz constant of each SOC layer can be slightly more than 1 and if we use a large number of SOC layers (e.g. 41 in LipConvnet-40), the Lipschitz constant of the full network $(\mathrm{Lip}(f))$ can be significantly larger than 1. To mitigate this issue, we take the following steps: (a) We use a large number of terms ($k = 15$) during test time to approximate the exponential which results in a small approximation error (using the error bound in Singla and Feizi [2021]) and (b) We compute $\mathrm{Lip}(f)$ by multiplying the lipschitz constant of all SOC layers (using the power method) and then divide the certificate by $\mathrm{Lip}(f)$.

Table 4: Provable robustness results on CIFAR-100 (LipConvnet-$5, 10$ results in Appendix Table 6)

LipConv	**Methods**	**Standard Accuracy**	**Provable Robust Accuracy**			**Increase**	
net-			36/255	72/255	108/255	(standard)	(36/255)
15	Baseline	48.06%	34.52%	23.08%	14.70%	_	_
	+ Fast	47.97%	33.84%	22.66%	14.26%	-0.09%	-0.68%
	+ CRC	**50.79%**	**37.50%**	**26.16%**	**17.27%**	**+2.73%**	**+2.98%**
20	Baseline	47.37%	33.99%	23.40%	14.69%	_	_
	+ Fast	46.41%	33.07%	22.06%	14.00%	-0.96%	-0.92%
	+ CRC	**51.84%**	**38.54%**	**27.32%**	**18.53%**	**+4.47%**	**+4.55%**
25	Baseline	45.77%	32.08%	21.36%	13.64%	_	_
	+ Fast	45.28%	31.67%	20.69%	13.26%	-0.49%	-0.41%
	+ CRC	**51.59%**	**39.27%**	**27.94%**	**19.06%**	**+5.82%**	**+7.19%**
30	Baseline	46.39%	33.08%	22.02%	13.77%	_	_
	+ Fast	45.86%	32.54%	21.18%	12.77%	-0.53%	-0.54%
	+ CRC	**50.97%**	**38.77%**	**27.73%**	**19.28%**	**+4.58%**	**+5.69%**
35	Baseline	43.42%	30.36%	19.71%	12.66%	_	_
	+ Fast	42.78%	29.88%	19.73%	12.52%	-0.64%	-0.48%
	+ CRC	**51.42%**	**39.01%**	**28.94%**	**20.29%**	**+8.00%**	**+8.65%**
40	Baseline	41.72%	28.53%	18.37%	11.49%	_	_
	+ Fast	42.07%	28.51%	18.86%	11.89%	+0.35%	-0.02%
	+ CRC	**50.11%**	**38.69%**	**28.45%**	**20.05%**	**+8.39%**	**+10.16%**

7.1 Results using Faster gradient computation

We show the reduction in training time per epoch (in seconds) on CIFAR-10 in Table 1 and CIFAR-100 in Appendix Table 8. In both Tables, we observe that for deeper networks (≥ 25 layers), the reduction in time per epoch is $\approx 30\%$. The corresponding standard and provable robust accuracy numbers are given in Tables 3 (CIFAR-10) and 4 (CIFAR-100) in the row "**+ Fast**". From the columns "Increase (Standard)" and "Increase (36/255)", we observe that the performance is similar to the baseline across all network architectures. For deeper networks: $\text{LipConvnet-}35, 40$, we observe an increase in performance for CIFAR-10 and small decrease ($< 0.64\%$) for CIFAR-100.

7.2 Results using projection pooling and CRC-Lip

We observe that using CRC and projection pooling (row "**+ CRC**") leads to significant improvements in performance across all LipConvnet architectures. On CIFAR-10, we observe significant improvements in both the standard ($\geq 1.63\%$, column "Increase (standard)") and provable robust accuracy ($\geq 3.14\%$, column "Increase (36/255)") across all architectures. On CIFAR-100, we also observe significant improvements in the standard ($\geq 2.49\%$) and provable robust accuracy ($\geq 2.27\%$). For deeper networks (LipConvnet-$35, 40$), on CIFAR-10, we observe even more significant gains in the standard ($\geq 5.84\%$) and provable robust accuracy ($\geq 8.76\%$). Similarly on CIFAR-100, we observe gains of $\geq 8.00\%$ (standard) and $\geq 8.65\%$ (provable robust).

Our results establish a new state-of-the-art for both the standard and provable robust accuracy across all attack radii. In Table 3 (CIFAR-10), the best "Baseline" standard and provable robust accuracy values (at $36/255, 72/255, 108/255$) are 77.78% and 63.31%, 46.42%, 31.53% respectively. The best "**+ CRC**" values are 79.57% and 67.13%, 53.17%, 38.60%. This results in improvements of $+1.79\%$ and $+3.82\%, +6.75\%, +7.07\%$ respectively. Similarly, in Table 4 (CIFAR-100), the best "Baseline" standard and provable robust values are 48.06% and $34.52\%, 23.40\%, 14.70\%$. The best "**+ CRC**" values are 51.84% $(+3.78\%)$ and 39.27% $(+4.75\%)$, 27.94% $(+4.54\%)$, 20.29% $(+5.59\%)$.

8 Acknowledgements

This project was supported in part by NSF CAREER AWARD 1942230, a grant from NIST 60NANB20D134, HR001119S0026 (GARD), ONR YIP award N00014-22-1-2271, Army Grant No. W911NF2120076 and the NSF award CCF2212458.

References

C. Anil, J. Lucas, and R. B. Grosse. Sorting out lipschitz function approximation. In *ICML*, 2018.

R. Bunel, J. Lu, I. Turkaslan, P. H. S. Torr, P. Kohli, and M. P. Kumar. Branch and bound for piecewise linear neural network verification. *J. Mach. Learn. Res.*, 21:42:1–42:39, 2020.

X. Cao and N. Z. Gong. Mitigating evasion attacks to deep neural networks via region-based classification. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, ACSAC 2017, page 278–287, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450353458. doi: 10.1145/3134600.3134606. URL `https://doi.org/10.1145/3134600.3134606`.

M. Cissé, P. Bojanowski, E. Grave, Y. N. Dauphin, and N. Usunier. Parseval networks: Improving robustness to adversarial examples. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 854–863. PMLR, 2017. URL `http://proceedings.mlr.press/v70/cisse17a.html`.

J. M. Cohen, E. Rosenfeld, and J. Z. Kolter. Certified adversarial robustness via randomized smoothing. In *ICML*, 2019.

F. Croce, M. Andriushchenko, and M. Hein. Provable robustness of relu networks via maximization of linear regions. *AISTATS 2019*, 2019.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, Dec. 1989. ISSN 0932-4194. doi: 10.1007/BF02551274. URL `http://dx.doi.org/10.1007/BF02551274`.

C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia. Incorporating second-order functional knowledge for better option pricing. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000. URL `https://proceedings.neurips.cc/paper/2000/file/44968aece94f667e4095002d140b5896-Paper.pdf`.

K. Dvijotham*, R. Stanforth, S. Gowal, T. Mann, and P. Kohli. A dual approach to scalable verification of deep networks. In *Proceedings of the Thirty-Fourth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-18)*, Corvallis, Oregon, 2018. AUAI Press.

K. D. Dvijotham*, A. Raghunathan, J. Uesato, S. Dathathri, A. Kurakin, I. Goodfellow, P. Kohli, J. Steinhardt, and P. Liang. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. In *Advances in Neural Information Processing Systems*, pages –, 2020.

H. Gouk, E. Frank, B. Pfahringer, and M. J. Cree. Regularisation of neural networks by enforcing lipschitz continuity, 2020.

S. Gowal, K. Dvijotham*, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelovic, T. A. Mann, and P. Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. In *ICCV*, volume 9, pages –, 2019.

I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5767–5777. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/892c3b1c6dccd52936e27cbd0ff683d6-Paper.pdf`.

E. Hoogeboom, V. G. Satorras, J. Tomczak, and M. Welling. The convolution exponential and generalized sylvester flows. *ArXiv*, abs/2006.01910, 2020.

Y. Huang, H. Zhang, Y. Shi, J. Z. Kolter, and A. Anandkumar. Training certifiably robust neural networks with efficient local lipschitz bounds. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=FTt28RYj5Pc.

J.-H. Jacobsen, A. W. Smeulders, and E. Oyallon. i-revnet: Deep invertible networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HJsjkMb0Z.

B. Kiani, R. Balestriero, Y. Lecun, and S. Lloyd. projunn: efficient method for training deep networks with unitary matrices, 2022. URL https://arxiv.org/abs/2203.05483.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

A. Kumar, A. Levine, S. Feizi, and T. Goldstein. Certifying confidence via randomized smoothing. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 5165–5177. Curran Associates, Inc., 2020a. URL https://proceedings.neurips.cc/paper/2020/file/37aa5dfc44dddd0d19d4311e2c7a0240-Paper.pdf.

A. Kumar, A. Levine, T. Goldstein, and S. Feizi. Curse of dimensionality on randomized smoothing for certifiable robustness. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5458–5467. PMLR, 13–18 Jul 2020b. URL http://proceedings.mlr.press/v119/kumar20b.html.

M. Lécuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. K. K. Jana. Certified robustness to adversarial examples with differential privacy. In *IEEE S&P 2019*, 2018.

K. Leino and M. Fredrikson. Relaxing local robustness. In *Neural Information Processing Systems (NIPS)*, 2021.

K. Leino, Z. Wang, and M. Fredrikson. Globally-robust neural networks. In *International Conference on Machine Learning (ICML)*, 2021.

A. Levine, S. Singla, and S. Feizi. Certifiably robust interpretation in deep learning, 2019.

B. Li, C. Chen, W. Wang, and L. Carin. Certified adversarial robustness with additive noise. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alche-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 9464–9474. Curran Associates, Inc., 2019a. URL https://proceedings.neurips.cc/paper/2019/file/335cd1b90bfa4ee70b39d08a4ae0cf2d-Paper.pdf.

Q. Li, S. Haque, C. Anil, J. Lucas, R. Grosse, and J.-H. Jacobsen. Preventing gradient attenuation in lipschitz constrained convolutional networks. *Conference on Neural Information Processing Systems*, 2019b.

X. Liu, M. Cheng, H. Zhang, and C. Hsieh. Towards robust neural networks via random self-ensemble. In *ECCV*, 2018.

P. M. Long and H. Sedghi. Generalization bounds for deep convolutional neural networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1e_FpNFDr.

J. Lu and M. P. Kumar. Neural network branching for neural network verification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=B1evfa4tPB.

T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B1QRgziT-.

C. Müller, F. Serre, G. Singh, M. Püschel, and M. Vechev. Scaling polyhedral neural network verification on gpus. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems*, volume 3, pages 733–746, 2021. URL `https://proceedings.mlsys.org/paper/2021/file/ca46c1b9512a7a8315fa3c5a946e8265-Paper.pdf`.

A. D. Palma, H. Behl, R. R. Bunel, P. Torr, and M. P. Kumar. Scaling the convex barrier with active sets. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=uQf0y7LrlTR`.

H. Qian and M. N. Wegman. L2-nonexpansive neural networks. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ByxGSsR9FQ`.

A. Raghunathan, J. Steinhardt, and P. Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *NeurIPS*, 2018.

H. Salman, J. Li, I. Razenshteyn, P. Zhang, H. Zhang, S. Bubeck, and G. Yang. Provably robust deep learning via adversarially trained smoothed classifiers. 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, pages 11292–11303. Curran Associates, Inc., 2019a. URL `https://proceedings.neurips.cc/paper/2019/file/3a24b25a7b092a252166a1641ae953e7-Paper.pdf`.

H. Salman, G. Yang, H. Zhang, C.-J. Hsieh, and P. Zhang. A convex relaxation barrier to tight robustness verification of neural networks. 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., 2019b. URL `https://proceedings.neurips.cc/paper/2019/file/246a3c5544feb054f3ea718f61adfa16-Paper.pdf`.

H. Salman, M. Sun, G. Yang, A. Kapoor, and J. Z. Kolter. Denoised smoothing: A provable defense for pretrained classifiers. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

H. Salman, S. Jain, E. Wong, and A. Madry. Certified patch robustness via smoothed vision transformers. *CoRR*, abs/2110.07719, 2021. URL `https://arxiv.org/abs/2110.07719`.

H. Sedghi, V. Gupta, and P. M. Long. The singular values of convolutional layers. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJevYoA9Fm`.

G. Singh, M. Püschel, and M. Vechev. Fast polyhedra abstract domain. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, page 46–59, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346603. doi: 10.1145/3009837.3009885. URL `https://doi.org/10.1145/3009837.3009885`.

G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev. Fast and effective robustness certification. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018a. URL `https://proceedings.neurips.cc/paper/2018/file/f2f446980d8e971ef3da97af089481c3-Paper.pdf`.

G. Singh, T. Gehr, M. Mirman, M. Puschel, and M. T. Vechev. Fast and effective robustness certification. In *NeurIPS*, 2018b.

G. Singh, R. Ganvir, M. Püschel, and M. Vechev. Beyond the single neuron convex barrier for neural network certification. 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., 2019a. URL `https://proceedings.neurips.cc/paper/2019/file/0a9fdbb17feb6ccb7ec405cfb85222c4-Paper.pdf`.

G. Singh, T. Gehr, M. Püschel, and M. Vechev. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019b. doi: 10.1145/3290354. URL `https://doi.org/10.1145/3290354`.

G. Singh, T. Gehr, M. Püschel, and M. Vechev. Robustness certification with refinement. In *International Conference on Learning Representations*, 2019c. URL `https://openreview.net/forum?id=HJgeEh09KQ`.

H. Singh, M. P. Kumar, P. Torr, and K. D. Dvijotham. Overcoming the convex barrier for simplex inputs. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=JXREUkyHi7u`.

S. Singla and S. Feizi. Second-order provable defenses against adversarial attacks. In H. D. III and A. 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 `http://proceedings.mlr.press/v119/singla20a.html`.

S. Singla and S. Feizi. Skew orthogonal convolutions. In *ICML*, 2021. URL `https://arxiv.org/abs/2105.11417`.

S. Singla, S. Singla, and S. Feizi. Improved deterministic l2 robustness on CIFAR-10 and CIFAR-100. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=tD7eCtaSkR`.

A. Sinha, H. Namkoong, and J. Duchi. Certifiable distributional robustness with principled adversarial training. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=Hk6kPgZA-`.

J. Su, W. Byeon, and F. Huang. Scaling-up diverse orthogonal convolutional networks with a paraunitary framework. In *ICML*, 2022. URL `https://arxiv.org/abs/2106.09121`.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014. URL `http://arxiv.org/abs/1312.6199`.

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

D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry. Robustness may be at odds with accuracy. In *ICLR*, 2018.

Y. Tsuzuku, I. Sato, and M. Sugiyama. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. In *NeurIPS*, 2018.

C. Villani. Optimal transport, old and new, 2008.

S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=ahYIlRBeCFw`.

L. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, L. Daniel, D. Boning, and I. Dhillon. Towards fast computation of certified robustness for ReLU networks. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5276–5285. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/weng18a.html`.

E. Wong and Z. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5286–5295, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/wong18a.html`.

E. Wong, F. R. Schmidt, J. H. Metzen, and J. Z. Kolter. Scaling provable adversarial defenses. In *NeurIPS*, 2018.

L. Xiao, Y. Bahri, J. Sohl-Dickstein, S. Schoenholz, and J. Pennington. Dynamical isometry and a mean field theory of CNNs: How to train 10,000-layer vanilla convolutional neural networks. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5393–5402, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/xiao18a.html`.

T. Yu, J. Li, Y. CAI, and P. Li. Constructing orthogonal convolutions in an explicit manner. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=Zr5W2LSRhD`.

B. Zhang, T. Cai, Z. Lu, D. He, and L. Wang. Towards certifying linfinity robustness using neural networks with linfinity-dist neurons. In *ICML*, 2021.

B. Zhang, D. Jiang, D. He, and L. Wang. Boosting the certified robustness of l-infinity distance nets. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=Q76Y7wkiji`.

H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems (NIPS), arXiv preprint arXiv:1811.00866*, dec 2018.

H. Zhang, P. Zhang, and C.-J. Hsieh. Recurjac: An efficient recursive algorithm for bounding jacobian matrix of neural networks and its applications. In *AAAI Conference on Artificial Intelligence (AAAI), arXiv preprint arXiv:1810.11783*, dec 2019.

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] in Section 7, we discuss how CRC-Lipmight lead to slower certification.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (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]
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]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A] because it is expensive to run.
   (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? [N/A]
   (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...
   (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]

Appendix

A Proofs

A.1 Proof of Theorem 1

Since $\mathbf{A}$ is skew symmetric, $\forall\, j,k,\;\; \mathbf{A}_{j,k} = -\mathbf{A}_{k,j}$.
Thus, we focus on computing $\nabla_{\mathbf{A}_{j,k}}\, \ell$ for $j > k$.

A.1.1 Preliminaries

Using equation (1) in the main text, we can rewrite $\mathbf{z}$ as follows:

$$\mathbf{z} = \mathbf{b} + \sum_{i=0}^{\infty} \frac{\mathbf{z}^{(i)}}{i!}, \quad \text{where } \; \mathbf{z}^{(i)} = \mathbf{A}^i\mathbf{x}$$

Using the chain rule, $\nabla_{\mathbf{A}_{j,k}}\, \ell$ can be computed as follows:

$$\begin{aligned}
\nabla_{\mathbf{A}_{j,k}}\, \ell &= \sum_{i=0}^{\infty} \left(\nabla_{\mathbf{A}_{j,k}}\, \mathbf{z}^{(i)}\right)^T \left(\nabla_{\mathbf{z}^{(i)}}\, \ell\right), \qquad \text{where} \quad \nabla_{\mathbf{z}^{(i)}}\, \ell = \frac{\nabla_{\mathbf{z}}\, \ell}{i!} \\
\implies \nabla_{\mathbf{A}_{j,k}}\, \ell &= \sum_{i=0}^{\infty} \left(\nabla_{\mathbf{A}_{j,k}}\, \mathbf{z}^{(i)}\right)^T \left(\frac{\nabla_{\mathbf{z}}\, \ell}{i!}\right) = \left(\sum_{i=0}^{\infty} \frac{\nabla_{\mathbf{A}_{j,k}}\, \mathbf{z}^{(i)}}{i!}\right)^T \nabla_{\mathbf{z}}\, \ell
\end{aligned}$$

Moreover, since $\mathbf{z}^{(0)}$ is independent of $\mathbf{W}_{j,k}$, we have: $\nabla_{\mathbf{A}_{j,k}}\, \mathbf{z}^{(0)} = 0$ implying:

$$\nabla_{\mathbf{A}_{j,k}}\, \ell = \left(\sum_{i=1}^{\infty} \frac{\nabla_{\mathbf{A}_{j,k}}\, \mathbf{z}^{(i)}}{i!}\right)^T \nabla_{\mathbf{z}}\, \ell \tag{10}$$

To compute each $\nabla_{\mathbf{A}_{j,k}}\mathbf{z}^{(i)}$, consider a matrix $\mathbf{E}$ defined below:

$$\mathbf{E}_{p,q} = \begin{cases} +1 & p = j,\; q = k \\ -1 & p = k,\; q = j \\ 0 & otherwise \end{cases} \tag{11}$$

Now, let $\mathbf{e}^{(j)}$ denote a vector that satisfies:

$$(\mathbf{e}^{(j)})_l = \begin{cases} 1 & l = j \\ 0 & otherwise \end{cases} \tag{12}$$

Clearly, $\mathbf{E}$ can be rewritten as follows:

$$\mathbf{E} = \left(\mathbf{e}^{(j)}(\mathbf{e}^{(k)})^T - \mathbf{e}^{(k)}(\mathbf{e}^{(j)})^T\right) \tag{13}$$

A.1.2 Computing $\nabla_{\mathbf{A}_{j,k}}\mathbf{z}^{(i)}$

Since $\mathbf{z}^{(i)} = \mathbf{A}^i\mathbf{x}$, we can compute $\nabla_{\mathbf{A}_{j,k}}\mathbf{z}^{(i)}$ as follows:

$$\begin{aligned}
\nabla_{\mathbf{A}_{j,k}}\mathbf{z}^{(i)} &= \lim_{\epsilon\to 0} \frac{(\mathbf{A} + \epsilon\mathbf{E})^i\mathbf{x} - \mathbf{A}^i\mathbf{x}}{\epsilon} = \lim_{\epsilon\to 0} \frac{(\mathbf{A} + \epsilon\mathbf{E})^i - \mathbf{A}^i}{\epsilon}\mathbf{x} \\
&= \lim_{\epsilon\to 0} \frac{\mathbf{A}^i + \epsilon\sum_{l=0}^{i-1}\left(\mathbf{A}^l\right)\mathbf{E}\left(\mathbf{A}^{i-l-1}\right) + \epsilon^2(\text{higher order terms}) - \mathbf{A}^i}{\epsilon}\mathbf{x} \\
&= \sum_{l=0}^{i-1}\left(\mathbf{A}^l\right)\mathbf{E}\left(\mathbf{A}^{i-l-1}\right)\mathbf{x}
\end{aligned} \tag{14}$$

A.1.3 Rewriting $\nabla_{\mathbf{A}_{j,k}}\ \ell$

Using equations (10) and (14), we can simplify $\nabla_{\mathbf{A}_{j,k}}\ \ell$ as follows:

$$\begin{aligned}
\nabla_{\mathbf{A}_{j,k}}\ \ell &= \left(\sum_{i=1}^{\infty} \frac{\nabla_{\mathbf{A}_{j,k}}\ \mathbf{z}^{(i)}}{i!}\right)^T \nabla_{\mathbf{z}}\ \ell \\
&= \left(\sum_{i=1}^{\infty}\sum_{l=0}^{i-1} \frac{1}{i!}\left(\mathbf{A}^l\right)\mathbf{E}\left(\mathbf{A}^{i-l-1}\mathbf{x}\right)\right)^T \nabla_{\mathbf{z}}\ \ell \\
&= \left(\sum_{i=1}^{\infty}\left(\sum_{l=0}^{i-1} \frac{1}{i!}\left(\mathbf{A}^l\right)\mathbf{E}\left(\mathbf{A}^{i-l-1}\mathbf{x}\right)\right)^T\right) \nabla_{\mathbf{z}}\ \ell \\
&= \left(\sum_{i=1}^{\infty}\sum_{l=0}^{i-1} \frac{1}{i!}\left(\mathbf{A}^{i-l-1}\mathbf{x}\right)^T\mathbf{E}^T\left(\mathbf{A}^l\right)^T\right) \nabla_{\mathbf{z}}\ \ell \\
&= \left(\sum_{i=1}^{\infty}\sum_{l=0}^{i-1} \frac{1}{i!}\left(\mathbf{A}^{i-l-1}\mathbf{x}\right)^T\mathbf{E}^T\left(\mathbf{A}^T\right)^l\right) \nabla_{\mathbf{z}}\ \ell \\
&= \left(\sum_{i=1}^{\infty}\sum_{l=0}^{i-1} \frac{1}{i!}\left(\mathbf{A}^{i-l-1}\mathbf{x}\right)^T\mathbf{E}^T\left(-\mathbf{A}\right)^l\right) \nabla_{\mathbf{z}}\ \ell \\
&= \sum_{i\geq 1,\ l\leq i-1}\left(\frac{1}{i!}\left(\mathbf{A}^{i-l-1}\mathbf{x}\right)^T\mathbf{E}^T\left(\left(-\mathbf{A}\right)^l\nabla_{\mathbf{z}}\ \ell\right)\right) \\
&= \sum_{l\geq 0,\ i\geq l+1}\left(\frac{1}{i!}\left(\mathbf{A}^{i-l-1}\mathbf{x}\right)^T\mathbf{E}^T\left(\left(-\mathbf{A}\right)^l\nabla_{\mathbf{z}}\ \ell\right)\right)
\end{aligned}$$

Rewriting $m = i - l - 1$, we get:

$$\nabla_{\mathbf{A}_{j,k}}\ \ell = \sum_{l\geq 0,\ m\geq 0}\left(\frac{1}{(m+l+1)!}\left(\mathbf{A}^m\mathbf{x}\right)^T\mathbf{E}^T\left(\left(-\mathbf{A}\right)^l\nabla_{\mathbf{z}}\ \ell\right)\right)$$

Note that the above equation can also be rewritten in the following equivalent forms:

$$\nabla_{\mathbf{A}_{j,k}} = \sum_{l=0}^{\infty}\left(\sum_{m=0}^{\infty}\frac{\mathbf{A}^m\mathbf{x}}{(m+l+1)!}\right)^T\mathbf{E}^T\left(\left(-\mathbf{A}\right)^l\nabla_{\mathbf{z}}\ \ell\right) \tag{15}$$

$$\nabla_{\mathbf{A}_{j,k}} = \sum_{m=0}^{\infty}\left(\mathbf{A}^m\mathbf{x}\right)^T\mathbf{E}^T\left(\sum_{l=0}^{\infty}\frac{\left(-\mathbf{A}\right)^l\nabla_{\mathbf{z}}\ \ell}{(m+l+1)!}\right) \tag{16}$$

Let us now define the following quantities:

$$\mathbf{u}^{(l)} = \sum_{m=0}^{\infty}\frac{\mathbf{A}^m\mathbf{x}}{(m+l)!} \tag{17}$$

$$\mathbf{v}^{(m)} = \sum_{l=0}^{\infty}\frac{\left(-\mathbf{A}\right)^l\nabla_{\mathbf{z}}\ \ell}{(m+l)!} \tag{18}$$

Note that they are similar to equations (2) and (3) in the maintext.
Using equations (17) and (18), we can rewrite (15) and (16) as follows:

$$\nabla_{\mathbf{A}_{j,k}} = \sum_{l=1}^{\infty}\left(\mathbf{u}^{(l)}\right)^T\mathbf{E}^T\left(\left(-\mathbf{A}\right)^{l-1}\nabla_{\mathbf{z}}\ \ell\right) \tag{19}$$

$$\nabla_{\mathbf{A}_{j,k}} = \sum_{m=1}^{\infty}\left(\mathbf{A}^{m-1}\mathbf{x}\right)^T\mathbf{E}^T\mathbf{v}^{(m)} \tag{20}$$

Using the definition of $\mathbf{E}$ (equation (11)), we obtain the following equivalent expressions of $\nabla_{\mathbf{A}_{j,k}}\, \ell$:

$$\nabla_{\mathbf{A}_{j,k}}\, \ell = -\sum_{l=1}^{\infty} \left(\mathbf{u}_j^{(l)} \left((-\mathbf{A})^{l-1} \nabla_{\mathbf{z}}\, \ell \right)_k - \mathbf{u}_k^{(l)} \left((-\mathbf{A})^{l-1} \nabla_{\mathbf{z}}\, \ell \right)_j \right) \tag{21}$$

$$\nabla_{\mathbf{A}_{j,k}}\, \ell = -\sum_{m=1}^{\infty} \left(\left(\mathbf{A}^{m-1}\mathbf{x} \right)_j \mathbf{v}_k^{(m)} - \left(\mathbf{A}^{m-1}\mathbf{x} \right)_k \mathbf{v}_j^{(m)} \right) \tag{22}$$

Using the above equation, we obtain the following equivalent expressions for $\nabla\, \ell$:

$$\nabla_{\mathbf{A}}\, \ell = -\sum_{l=1}^{\infty} \left(\mathbf{u}^{(l)} \left((-\mathbf{A})^{l-1} \nabla_{\mathbf{z}}\, \ell \right)^T - \left((-\mathbf{A})^{l-1} \nabla_{\mathbf{z}}\, \ell \right) \left(\mathbf{u}^{(l)} \right)^T \right) \tag{23}$$

$$\nabla_{\mathbf{A}}\, \ell = -\sum_{m=1}^{\infty} \left(\left(\mathbf{A}^{m-1}\mathbf{x} \right) \left(\mathbf{v}^{(m)} \right)^T - \mathbf{v}^{(m)} \left(\mathbf{A}^{m-1}\mathbf{x} \right)^T \right) \tag{24}$$

A.2 Proof of Proposition 1

Consider the inequality given in equation (6):

$$\min_{i \neq l} \min_{h_l(\mathbf{y}^*) = h_i(\mathbf{y}^*)} \|\mathbf{y}^* - g(\mathbf{x})\|_2 \ \geq R$$

The above inequality says that for the prediction of h to be different from l for some input $\mathbf{y}^*$, the l_2 distance $\|\mathbf{y}^* - g(\mathbf{x})\|_2$ must be at least R.
We first take the contrapositive of this statement:

$$\|\mathbf{y}^* - g(\mathbf{x})\|_2 \ \leq R \implies \forall_i\ h_l(\mathbf{y}^*) \geq h_i(\mathbf{y}^*) \tag{25}$$

But since g is 1-Lipschitz, we have:

$$\|\mathbf{x}^* - \mathbf{x}\|_2 \leq R \implies \|g(\mathbf{x}^*) - g(\mathbf{x})\|_2 \leq R \tag{26}$$

Using both (25) and (26), we get:

$$\|\mathbf{x}^* - \mathbf{x}\|_2 \leq R \implies \forall_i\ h_l(g(\mathbf{x}^*)) \geq h_i(g(\mathbf{x}^*)) \implies \forall_i\ f_l(\mathbf{x}^*) \geq f_i(\mathbf{x}^*)$$

Thus, in an l_2 ball of radius R, prediction l remains unchanged and the function is provably robust.

A.3 Proof of Theorem 2

Let $\mathbf{x}^*,\ \mathbf{y}^* \in \mathcal{M}$ so that $d_{\mathcal{M}}(\mathbf{x})$ and $d_{\mathcal{M}}(\mathbf{y})$ satisfy:

$$d_{\mathcal{M}}(\mathbf{x}) = \|\mathbf{x}^* - \mathbf{x}\|_2 \qquad d_{\mathcal{M}}(\mathbf{y}) = \|\mathbf{y}^* - \mathbf{y}\|_2 \tag{27}$$

Since $\mathbf{x}^*$ and $\mathbf{y}^*$ minimize the l_2 distance of $\mathbf{x}$ and $\mathbf{y}$ respectively to $\mathcal{M}$, we have:

$$\|\mathbf{x}^* - \mathbf{x}\|_2 \leq \|\mathbf{y}^* - \mathbf{x}\|_2, \qquad \|\mathbf{y}^* - \mathbf{y}\|_2 \leq \|\mathbf{x}^* - \mathbf{y}\|_2 \tag{28}$$

Using the triangle inequality for the expressions $\|\mathbf{y}^* - \mathbf{x}\|_2$ and $\|\mathbf{x}^* - \mathbf{y}\|_2$, we have:

$$\|\mathbf{y}^* - \mathbf{x}\|_2 \leq \|\mathbf{y}^* - \mathbf{y}\|_2 + \|\mathbf{y} - \mathbf{x}\|_2, \qquad \|\mathbf{x}^* - \mathbf{y}\|_2 \leq \|\mathbf{x}^* - \mathbf{x}\|_2 + \|\mathbf{x} - \mathbf{y}\|_2 \tag{29}$$

Using inequalities (28) and (29), we get:

$$\|\mathbf{x}^* - \mathbf{x}\|_2 \leq \|\mathbf{y}^* - \mathbf{y}\|_2 + \|\mathbf{y} - \mathbf{x}\|_2, \qquad \|\mathbf{y}^* - \mathbf{y}\|_2 \leq \|\mathbf{x}^* - \mathbf{x}\|_2 + \|\mathbf{x} - \mathbf{y}\|_2 \tag{30}$$

But then using equation (27) and inequality (30), we get:

$$\begin{aligned} & d_{\mathcal{M}}(\mathbf{x}) \leq d_{\mathcal{M}}(\mathbf{y}) + \|\mathbf{y} - \mathbf{x}\|_2, \qquad d_{\mathcal{M}}(\mathbf{y}) \leq d_{\mathcal{M}}(\mathbf{x}) + \|\mathbf{x} - \mathbf{y}\|_2 \\ & d_{\mathcal{M}}(\mathbf{x}) - d_{\mathcal{M}}(\mathbf{y}) \leq \|\mathbf{y} - \mathbf{x}\|_2, \qquad d_{\mathcal{M}}(\mathbf{y}) - d_{\mathcal{M}}(\mathbf{x}) \leq \|\mathbf{x} - \mathbf{y}\|_2 \\ & \implies |d_{\mathcal{M}}(\mathbf{x}) - d_{\mathcal{M}}(\mathbf{y})| \leq \|\mathbf{x} - \mathbf{y}\|_2 \end{aligned}$$

A.4 Proof of Corollary 1

We define a function $d_{\mathcal{M}}(\mathbf{x})$ that is (without loss of generality) positive in $\mathcal{A}$ and negative in $\mathcal{B}$:

$$d_{\mathcal{M}}(\mathbf{x}) = \begin{cases} \mathbf{x} \in \mathcal{A} & +\min_{\mathbf{x}^* \in \mathcal{M}} \|\mathbf{x}^* - \mathbf{x}\|_2 \\ \mathbf{x} \in \mathcal{B} & -\min_{\mathbf{x}^* \in \mathcal{M}} \|\mathbf{x}^* - \mathbf{x}\|_2 \end{cases}$$

Using Theorem 2, it is easy to see that within each set $\mathcal{A}$ and $\mathcal{B}$, $d_{\mathcal{M}}$ is 1-Lipschitz and continuous. Similarly, because every line segment connecting $\mathcal{A}$ and $\mathcal{B}$ crosses through some point on the $\mathcal{M}$, it is evident that $d_{\mathcal{M}}$ is continuous on each path connecting $\mathcal{A}$ to $\mathcal{B}$.
So, we next prove that $d_{\mathcal{M}}$ is 1-Lipschitz if $\mathbf{x} \in \mathcal{A}$ and $\mathbf{y} \in \mathcal{B}$ i.e.:

$$|d_{\mathcal{M}}(\mathbf{x}) - d_{\mathcal{M}}(\mathbf{y})| \le \|\mathbf{x} - \mathbf{y}\|_2$$

Let $\mathbf{p} \in \mathcal{M}$ be a point on the line segment connecting $\mathbf{x}$ and $\mathbf{y}$ such that:

$$\|\mathbf{x} - \mathbf{y}\|_2 = \|\mathbf{x} - \mathbf{p}\|_2 + \|\mathbf{p} - \mathbf{y}\|_2 \tag{31}$$

But then by the definitions of $d_{\mathcal{M}}(\mathbf{x})$ and $d_{\mathcal{M}}(\mathbf{y})$:

$$|d_{\mathcal{M}}(\mathbf{x})| \le \|\mathbf{x} - \mathbf{p}\|_2, \qquad |d_{\mathcal{M}}(\mathbf{y})| \le \|\mathbf{p} - \mathbf{y}\|_2 \tag{32}$$

Using equation (31) and inequality (32), we get:

$$\begin{aligned} &|d_{\mathcal{M}}(\mathbf{x})| + |d_{\mathcal{M}}(\mathbf{y})| \le \|\mathbf{x} - \mathbf{p}\|_2 + \|\mathbf{p} - \mathbf{y}\|_2 = \|\mathbf{x} - \mathbf{y}\|_2 \\ \implies &|d_{\mathcal{M}}(\mathbf{x})| + |d_{\mathcal{M}}(\mathbf{y})| \le \|\mathbf{x} - \mathbf{y}\|_2 \end{aligned} \tag{33}$$

Since $d_{\mathcal{M}}(\mathbf{x})$ and $d_{\mathcal{M}}(\mathbf{y})$ have opposite signs, we have:

$$|d_{\mathcal{M}}(\mathbf{x})| + |d_{\mathcal{M}}(\mathbf{y})| = |d_{\mathcal{M}}(\mathbf{x}) - d_{\mathcal{M}}(\mathbf{y})| \tag{34}$$

Using inequality (33) and equation (34), we prove:

$$|d_{\mathcal{M}}(\mathbf{x}) - d_{\mathcal{M}}(\mathbf{y})| \le \|\mathbf{x} - \mathbf{y}\|_2$$

Hence, $|d_{\mathcal{M}}(\mathbf{x}) - d_{\mathcal{M}}(\mathbf{y})|$ is 1-Lipschitz.

B Approximating the convolution weight gradient

B.1 Simplifying the expression for $\nabla_{\mathbf{A}_{j,k}} \ell$

Recall the expression for $\nabla_{\mathbf{A}_{j,k}} \ell$ using equations (10) and (14):

$$\nabla_{\mathbf{A}_{j,k}} \ell = \left(\sum_{i=1}^{\infty} \frac{\nabla_{\mathbf{A}_{j,k}} \mathbf{z}^{(i)}}{i!} \right)^T \nabla_{\mathbf{z}} \ell, \qquad \text{where } \nabla_{\mathbf{A}_{j,k}} \mathbf{z}^{(i)} = \sum_{l=0}^{i-1} \left(\mathbf{A}^l\right) \mathbf{E} \left(\mathbf{A}^{i-l-1}\right) \mathbf{x}$$

We replace $\nabla_{\mathbf{A}_{j,k}} \mathbf{z}^{(i)}$ in $\nabla_{\mathbf{A}_{j,k}} \ell$ and take $\mathbf{x}$ out of the transpose:

$$\begin{aligned} \nabla_{\mathbf{A}_{j,k}} \ell &= \mathbf{x}^T \left(\sum_{i=1}^{\infty} \frac{1}{i!} \sum_{l=0}^{i-1} \left(\mathbf{A}^l\right) \mathbf{E} \left(\mathbf{A}^{i-l-1}\right) \right)^T \nabla_{\mathbf{z}} \ell \\ &= \mathbf{x}^T \left(\mathbf{E} + \frac{1}{2!} \left(\mathbf{EA} + \mathbf{AE}\right) + \frac{1}{3!} \left(\mathbf{EA}^2 + \mathbf{AEA} + \mathbf{A}^2\mathbf{E}\right) + \dots \right)^T \nabla_{\mathbf{z}} \ell \end{aligned}$$

We simplify the above as follows:

$$\nabla_{\mathbf{A}_{j,k}} \ell \tag{35}$$

$$= \mathbf{x}^T \left(\mathbf{E} + \frac{1}{2!} \left(\mathbf{EA} + \mathbf{AE}\right) + \frac{1}{3!} \left(\mathbf{EA}^2 + \mathbf{A}^2\mathbf{E}\right) + \frac{1}{4!} \left(\mathbf{EA}^3 + \mathbf{A}^3\mathbf{E}\right) + \dots \right)^T \nabla_{\mathbf{z}} \ell \tag{36}$$

$$+ \mathbf{x}^T \mathbf{R}^T \left(\nabla_{\mathbf{z}} \ell\right) \tag{37}$$

where $\mathbf{R}$ is defined as follows:

$$\mathbf{R} = \frac{1}{3!} \left(\mathbf{AEA}\right) + \frac{1}{4!} \left(\mathbf{AEA}^2 + \mathbf{A}^2\mathbf{EA}\right) + \frac{1}{5!} \left(\mathbf{AEA}^3 + \mathbf{A}^2\mathbf{EA}^2 + \mathbf{A}^3\mathbf{EA}\right) + \dots \tag{38}$$

In the above simplification, $\mathbf{R}$ contains all terms of the form $\mathbf{A}^l\mathbf{EA}^m$ where both $l \ge 1$ and $m \ge 1$.

B.2 Simplifying the expression for $\left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T \mathbf{v}^{(1)}$

Recall that $\mathbf{u}^{(1)}, \mathbf{v}^{(1)}$ are given by (equations (17) and (18)):

$$\mathbf{u}^{(1)} = \left(\sum_{l=1}^{\infty} \frac{\mathbf{A}^{l-1}}{l!} \right) \mathbf{x}, \qquad \mathbf{v}^{(1)} = \left(\sum_{m=1}^{\infty} \frac{(-\mathbf{A})^{m-1}}{m!} \right) \nabla_{\mathbf{z}}\, \ell \tag{39}$$

We first replace $\mathbf{v}^{(1)}$ using the above in $\left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T \mathbf{v}^{(1)}$:

$$\left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T \mathbf{v}^{(1)} = \left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T \left(\sum_{m=1}^{\infty} \frac{(-\mathbf{A})^{m-1}}{m!} \right) \nabla_{\mathbf{z}}\, \ell \tag{40}$$

Given 2 matrices $\mathbf{B}, \mathbf{C}$, we have $\left(\mathbf{B}^T\right)^i = \left(\mathbf{B}^i\right)^T$ and $(\mathbf{B}+\mathbf{C})^T = \mathbf{B}^T + \mathbf{C}^T$. Thus:

$$\left(\sum_{m=1}^{\infty} \frac{\mathbf{A}^{m-1}}{m!} \right)^T = \left(\sum_{m=1}^{\infty} \frac{(-\mathbf{A})^{m-1}}{m!} \right)$$

Replacing the above in equation (40), we get:

$$\left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T \mathbf{v}^{(1)} = \left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T \left(\sum_{m=1}^{\infty} \frac{\mathbf{A}^{m-1}}{m!} \right)^T \nabla_{\mathbf{z}}\, \ell$$

Given 3 matrices, $\mathbf{B}, \mathbf{C}, \mathbf{D}$, we have $\mathbf{D}^T\mathbf{C}^T\mathbf{B}^T = (\mathbf{BCD})^T$. Thus:

$$\left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T \mathbf{v}^{(1)} = \left(\left(\sum_{m=1}^{\infty} \frac{\mathbf{A}^{m-1}}{m!} \right) \mathbf{E}\mathbf{u}^{(1)} \right)^T \nabla_{\mathbf{z}}\, \ell$$

Now, we replace $\mathbf{u}^{(1)}$ using equation (39) and take $\mathbf{x}$ out of the transpose:

$$\left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T \mathbf{v}^{(1)} = \mathbf{x}^T \left(\left(\sum_{m=1}^{\infty} \frac{\mathbf{A}^{m-1}}{m!} \right) \mathbf{E} \left(\sum_{l=1}^{\infty} \frac{\mathbf{A}^{l-1}}{l!} \right) \right)^T \nabla_{\mathbf{z}}\, \ell$$

Expanding the two series, we get:

$$\left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T \mathbf{v}^{(1)} = \mathbf{x}^T \left(\left(\mathbf{I} + \frac{\mathbf{A}}{2!} + \frac{\mathbf{A}^2}{3!} + \dots \right) \mathbf{E} \left(\mathbf{I} + \frac{\mathbf{A}}{2!} + \frac{\mathbf{A}^2}{3!} + \dots \right) \right)^T \nabla_{\mathbf{z}}\, \ell$$

We simplify the above as follows:

$$\left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T \mathbf{v}^{(1)} \tag{41}$$

$$= \mathbf{x}^T \left(\mathbf{E} + \frac{1}{2!}\left(\mathbf{EA} + \mathbf{AE}\right) + \frac{1}{3!}\left(\mathbf{EA}^2 + \mathbf{A}^2\mathbf{E}\right) + \frac{1}{4!}\left(\mathbf{EA}^3 + \mathbf{A}^3\mathbf{E}\right) + \dots \right)^T \nabla_{\mathbf{z}}\, \ell \tag{42}$$

$$+ \mathbf{x}^T\mathbf{S}^T \left(\nabla_{\mathbf{z}}\, \ell\right) \tag{43}$$

where $\mathbf{S}$ is defined as follows:

$$\mathbf{S} = \left(\frac{\mathbf{AEA}}{2!2!} \right) + \left(\frac{\mathbf{AEA}^2}{2!3!} + \frac{\mathbf{A}^2\mathbf{EA}}{3!2!} \right) + \left(\frac{\mathbf{AEA}^3}{2!4!} + \frac{\mathbf{A}^2\mathbf{EA}^2}{3!3!} + \frac{\mathbf{A}^3\mathbf{EA}}{4!2!} \right) + \dots \tag{44}$$

B.3 Comparing the two expressions

Comparing equations (35) and (41), the first part in the two are same.
The first part are the equations (36) and (42) respectively.
The two expressions only differ in the terms $\mathbf{x}^T\mathbf{R}^T\left(\nabla_{\mathbf{z}}\, \ell\right)$ and $\mathbf{x}^T\mathbf{S}^T\left(\nabla_{\mathbf{z}}\, \ell\right)$.
Note that in $\mathbf{R}$ (equation (38)), the term $\mathbf{A}^l\mathbf{E}\mathbf{A}^m$ is multiplied by $1/(l+m+1)!$.
And in $\mathbf{S}$ (equation (44)), the term $\mathbf{A}^l\mathbf{E}\mathbf{A}^m$ is multiplied by $1/(l+1)!(m+1)!$.

Thus, since the quantities $\mathbf{R}$ and $\mathbf{S}$ are multiplied by factorial in the denominator.
For higher values of l or m, both the expressions converge to 0.
Moreover, $\mathbf{S}-\mathbf{R}$ for smaller values of l, m can be directly computed using equations (38) and (44):

$$\mathbf{S}-\mathbf{R} = \left(\frac{\mathbf{AEA}}{12}\right) + \left(\frac{\mathbf{AEA}^2}{24} + \frac{\mathbf{A}^2\mathbf{EA}}{24}\right) + \left(\frac{\mathbf{AEA}^3}{80} + \frac{\mathbf{A}^2\mathbf{EA}^2}{360/7} + \frac{\mathbf{A}^3\mathbf{EA}}{80}\right) + \dots \quad (45)$$

We can see that the denominator in the above terms are already very small even for $l+m=4$.
Thus, we simply assume $\mathbf{S}-\mathbf{R} \approx \mathbf{0}$.
This results in the following approximate expression for $\nabla_{\mathbf{A}_{j,k}} \ell$:

$$\nabla_{\mathbf{A}_{j,k}} \ell \approx \left(\mathbf{u}^{(1)}\right)^T \mathbf{E}^T \mathbf{v}^{(1)} \quad (46)$$

Using the definition of $\mathbf{E}$ (equation (11)), we obtain the following expression of $\nabla_{\mathbf{A}_{j,k}} \ell$:

$$\nabla_{\mathbf{A}_{j,k}} \ell \approx -(\mathbf{u}_j^{(1)}\mathbf{v}_k^{(1)} - \mathbf{u}_k^{(1)}\mathbf{v}_j^{(1)}) \quad (47)$$

Using equation (47), the gradient with respect to matrix $\mathbf{A}$ can be computed as follows:

$$\nabla_{\mathbf{A}} \ell \approx -\left(\mathbf{u}^{(1)}\left(\mathbf{v}^{(1)}\right)^T - \mathbf{v}^{(1)}\left(\mathbf{u}^{(1)}\right)^T\right) \quad (48)$$

C Iterations for $\mathbf{z}$ and $\nabla_{\mathbf{z}} \ell$

Recall that we use the following iterations to compute $\mathbf{z}$ during the forward pass:

$$\mathbf{u}^{(i)} = \begin{cases} \mathbf{x} & i = k-1 \\ \mathbf{x} + \left(\mathbf{A}\mathbf{u}^{(i+1)}\right)/(i+1) & i \leq k-2 \end{cases}$$

We have to prove that $\forall\ k,\ \mathbf{u}^{(0)} = \mathbf{z}$ where:

$$\mathbf{z} = \sum_{i=0}^{k-1} \frac{\mathbf{A}^i\mathbf{x}}{i!}$$

To prove the same, we use induction over k.
For $k=1$, $\mathbf{u}^{(0)} = \mathbf{z} = \mathbf{x}$ by definition.
For $k \geq 2$, we assume that the statement is true for $j \leq k-1$ and try to prove for k.
For k, we have the following expressions for $\mathbf{u}^{(k-1)}$ and $\mathbf{u}^{(k-2)}$:

$$\mathbf{u}^{(k-1)} = \mathbf{x}, \quad \mathbf{u}^{(k-2)} = \mathbf{x} + \mathbf{A}\mathbf{x}/(k-1)$$

Next, we apply the iteration on $\mathbf{u}^{(k-2)}$ till $\mathbf{u}^{(0)}$.
By induction, the first term i.e. $\mathbf{x}$ simply results in $\sum_{i=0}^{k-2}(\mathbf{A}^i\mathbf{x})/i!$.
The second term i.e. $\mathbf{A}\mathbf{x}/k$ in $(\mathbf{A}^{k-1}\mathbf{x})/(k-1)!$. Thus, we have:

$$\mathbf{u}^{(0)} = \left(\sum_{i=0}^{k-2} \frac{\mathbf{A}^i\mathbf{x}}{i!}\right) + \frac{\mathbf{A}^{k-1}\mathbf{x}}{(k-1)!} = \sum_{i=0}^{k-1} \frac{\mathbf{A}^i\mathbf{x}}{i!}$$

The same can proof follows for $\nabla_{\mathbf{z}} \ell$ except that we replace $\mathbf{A}$ with $-\mathbf{A}$.

D Review of Curvature-based Robustness Certificate

Given a binary classifier $f : \mathbb{R}^d \to \mathbb{R}$ and input $\mathbf{x}^{(0)}$, we are interested in finding its shortest l_2 distance to the decision boundary: $\min_{\mathbf{x}} \|\mathbf{x}-\mathbf{x}^{(0)}\|_2$ where $f(\mathbf{x}) = 0$. Using lagrange multiplier η for the constraint $f(\mathbf{x}) = 0$, the *primal* and *dual* problems can be written as follows:

$$\textit{(primal)} \quad \min_{\mathbf{x}} \max_{\eta} \left[\frac{1}{2}\|\mathbf{x}-\mathbf{x}^{(0)}\|_2^2 + \eta f(\mathbf{x})\right] \geq \max_{\eta} \min_{\mathbf{x}} \left[\frac{1}{2}\|\mathbf{x}-\mathbf{x}^{(0)}\|_2^2 + \eta f(\mathbf{x})\right] \quad \textit{(dual)}$$

Singla and Feizi [2020] prove that: If $m\mathbf{I} \preccurlyeq \nabla_{\mathbf{x}}^2 f \preccurlyeq M\mathbf{I}$, the *dual* minimization can be solved using convex optimization for $-1/M \leq \eta \leq -1/m$.
Furthermore, they show that if the solution $\mathbf{x}$ of the solution satisfies $f(\mathbf{x}) = 0$, the certificate is tight i.e. *primal* = *dual*.

E Complete results on CIFAR-10 and CIFAR-100

Table 5: Results for provable robustness against adversarial examples on the CIFAR-10 dataset.

LipConv	**Methods**	**Standard**	**Provable Robust Accuracy**			**Increase**	
net-		**Accuracy**	36/255	72/255	108/255	(standard)	(36/255)
5	Baseline	76.68%	60.09%	42.89%	27.45%	_	_
	+ Fast	76.03%	60.12%	42.90%	27.91%	-0.65%	+0.03%
	+ Pool	77.53%	62.17%	44.94%	29.74%	+0.85%	+2.08%
	+ CRC	**79.36%**	**67.13%**	**52.49%**	**38.19%**	**+2.68%**	**+7.04%**
10	Baseline	77.73%	62.82%	45.62%	30.09%	_	_
	+ Fast	77.76%	62.42%	45.48%	30.38%	+0.03%	-0.40%
	+ Pool	77.31%	62.02%	46.04%	30.99%	-0.42%	-0.80%
	+ CRC	**79.57%**	**66.75%**	**53.17%**	**38.60%**	**+1.84%**	**+3.93%**
15	Baseline	77.78%	62.75%	46.34%	31.38%	_	_
	+ Fast	77.75%	62.52%	46.23%	31.19%	-0.03%	-0.23%
	+ Pool	77.75%	63.14%	47.01%	32.05%	-0.03%	+0.39%
	+ CRC	**79.44%**	**66.99%**	**52.56%**	**38.30%**	**+1.66%**	**+4.24%**
20	Baseline	77.50%	63.31%	46.42%	31.53%	_	_
	+ Fast	77.13%	62.05%	45.86%	31.13%	-0.37%	-1.26%
	+ Pool	77.61%	63.25%	47.10%	32.84%	+0.11%	-0.06%
	+ CRC	**79.13%**	**66.45%**	**52.45%**	**38.12%**	**+1.63%**	**+3.14%**
25	Baseline	77.18%	62.46%	45.78%	31.16%	_	_
	+ Fast	76.94%	61.91%	45.59%	30.69%	-0.24%	-0.55%
	+ Pool	75.44%	60.13%	45.02%	30.44%	-1.74%	-2.33%
	+ CRC	**79.19%**	**66.28%**	**51.74%**	**37.99%**	**+2.01%**	**+3.82%**
30	Baseline	74.43%	59.65%	43.76%	29.16%	_	_
	+ Fast	74.69%	58.84%	43.33%	28.93%	+0.26%	-0.81%
	+ Pool	74.12%	59.17%	42.76%	28.75%	-0.31%	-0.48%
	+ CRC	**78.64%**	**66.05%**	**51.31%**	**37.30%**	**+4.21%**	**+6.40%**
35	Baseline	72.73%	57.18%	42.08%	28.09%	_	_
	+ Fast	72.91%	57.58%	41.52%	27.37%	+0.18%	+0.40%
	+ Pool	72.84%	56.71%	40.41%	26.62%	+0.11%	-0.47%
	+ CRC	**78.57%**	**65.94%**	**52.04%**	**37.63%**	**+5.84%**	**+8.76%**
40	Baseline	71.33%	55.74%	39.32%	26.06%	_	_
	+ Fast	71.60%	56.15%	39.82%	25.63%	+0.27%	+0.41%
	+ Pool	70.19%	54.22%	37.51%	23.88%	-1.14%	-1.52%
	+ CRC	**78.41%**	**65.51%**	**51.32%**	**37.30%**	**+7.08%**	**+9.77%**

Table 6: Results for provable robustness against adversarial examples on the CIFAR-100 dataset.

LipConv	**Methods**	**Standard**	**Provable Robust Accuracy**			**Increase**	
net-		**Accuracy**	36/255	72/255	108/255	(standard)	(36/255)
5	Baseline	46.74%	32.45%	21.04%	12.88%	_	_
	+ Fast	46.18%	32.15%	20.79%	12.87%	-0.56%	-0.30%
	+ Pool	47.47%	33.51%	22.06%	13.73%	+0.73%	+1.06%
	+ CRC	**49.73%**	**36.05%**	**24.68%**	**16.31%**	**+2.99%**	**+3.60%**
10	Baseline	47.96%	34.33%	22.55%	14.23%	_	_
	+ Fast	47.24%	33.79%	22.38%	14.36%	-0.72%	-0.54%
	+ Pool	47.32%	34.01%	22.28%	14.04%	-0.64%	-0.32%
	+ CRC	**50.45%**	**36.60%**	**25.57%**	**16.93%**	**+2.49%**	**+2.27%**
15	Baseline	48.06%	34.52%	23.08%	14.70%	_	_
	+ Fast	47.97%	33.84%	22.66%	14.26%	-0.09%	-0.68%
	+ Pool	47.44%	34.12%	22.59%	14.48%	-0.62%	-0.40%
	+ CRC	**50.79%**	**37.50%**	**26.16%**	**17.27%**	**+2.73%**	**+2.98%**
20	Baseline	47.37%	33.99%	23.40%	14.69%	_	_
	+ Fast	46.41%	33.07%	22.06%	14.00%	-0.96%	-0.92%
	+ Pool	46.54%	32.75%	21.64%	13.22%	-0.83%	-1.24%
	+ CRC	**51.84%**	**38.54%**	**27.32%**	**18.53%**	**+4.47%**	**+4.55%**
25	Baseline	45.77%	32.08%	21.36%	13.64%	_	_
	+ Fast	45.28%	31.67%	20.69%	13.26%	-0.49%	-0.41%
	+ Pool	45.26%	31.87%	20.88%	13.65%	-0.51%	-0.21%
	+ CRC	**51.59%**	**39.27%**	**27.94%**	**19.06%**	**+5.82%**	**+7.19%**
30	Baseline	46.39%	33.08%	22.02%	13.77%	_	_
	+ Fast	45.86%	32.54%	21.18%	12.77%	-0.53%	-0.54%
	+ Pool	45.15%	31.53%	21.18%	13.14%	-1.24%	-1.55%
	+ CRC	**50.97%**	**38.77%**	**27.73%**	**19.28%**	**+4.58%**	**+5.69%**
35	Baseline	43.42%	30.36%	19.71%	12.66%	_	_
	+ Fast	42.78%	29.88%	19.73%	12.52%	-0.64%	-0.48%
	+ Pool	42.33%	28.65%	18.59%	11.75%	-1.09%	-1.71%
	+ CRC	**51.42%**	**39.01%**	**28.94%**	**20.29%**	**+8.00%**	**+8.65%**
40	Baseline	41.72%	28.53%	18.37%	11.49%	_	_
	+ Fast	42.07%	28.51%	18.86%	11.89%	+0.35%	-0.02%
	+ Pool	40.27%	27.09%	17.36%	10.62%	-1.45%	-1.44%
	+ CRC	**50.11%**	**38.69%**	**28.45%**	**20.05%**	**+8.39%**	**+10.16%**

F Running time comparison for Faster SOC gradient computation

Table 7: Results on CIFAR-10

Architecture	**Time per epoch (in seconds)**		**Reduction in**
	Ours	Previous	**Time per epoch**
LipConvnet-5	**25.343**	30.633	**-17.27%**
LipConvnet-10	**37.108**	48.436	**-23.39%**
LipConvnet-15	**49.274**	66.741	**-26.17%**
LipConvnet-20	**61.586**	83.183	**-25.96%**
LipConvnet-25	**71.510**	100.700	**-28.99%**
LipConvnet-30	**83.996**	119.549	**-29.74%**
LipConvnet-35	**95.065**	137.100	**-30.66%**
LipConvnet-40	**106.007**	156.254	**-32.16%**

Table 8: Results on CIFAR-100

Architecture	**Time per epoch (in seconds)**		**Reduction in**
	Ours	Previous	**Time per epoch**
LipConvnet-5	**25.034**	31.877	**-21.47%**
LipConvnet-10	**36.872**	49.446	**-25.43%**
LipConvnet-15	**48.769**	68.071	**-28.36%**
LipConvnet-20	**60.416**	85.007	**-28.93%**
LipConvnet-25	**71.064**	101.933	**-30.28%**
LipConvnet-30	**86.319**	122.774	**-29.69%**
LipConvnet-35	**97.802**	140.606	**-30.44%**
LipConvnet-40	**110.345**	157.503	**-29.94%**

G Example of projection pooling layer

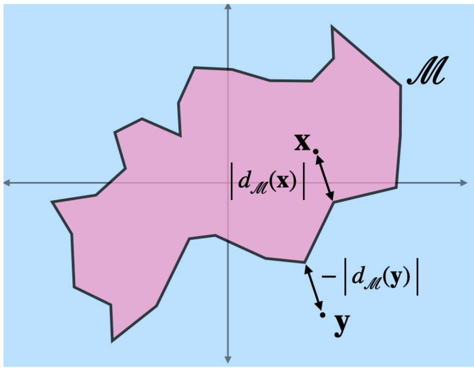

$\mathscr{M}$
$\mathbf{x}$
$\left|d_{\mathscr{M}}(\mathbf{x})\right|$
$-\left|d_{\mathscr{M}}(\mathbf{y})\right|$
$\mathbf{y}$

Figure 2: Example 2D projection pooling using piecewise linear curve