On Certifying Non-uniform Bounds against Adversarial Attacks

Chen Liu [1] **Ryota Tomioka** [2] **Volkan Cevher** [1]

Abstract

This work studies the robustness certification problem of neural network models, which aims to find certified adversary-free regions as large as possible around data points. In contrast to the existing approaches that seek regions bounded uniformly along all input features, we consider non-uniform bounds and use it to study the decision boundary of neural network models. We formulate our target as an optimization problem with nonlinear constraints. Then, a framework applicable for general feedforward neural networks is proposed to bound the output logits so that the relaxed problem can be solved by the augmented Lagrangian method. Our experiments show the non-uniform bounds have larger volumes than uniform ones and the geometric similarity of the non-uniform bounds gives a quantitative, data-agnostic metric of input features' robustness. Further, compared with normal models, the robust models have even larger non-uniform bounds and better interpretability.

1. Introduction

Although deep neural networks have achieved great success and state-of-the-art performances in many tasks, they are vulnerable to some adversarial attacks of the input data (Szegedy et al., 2013; Goodfellow et al., 2015; Moosavi-Dezfooli et al., 2017). This issue can be contextualized as a game between the attackers who try to adversarially manipulate the input data and the defenders who try to obtain the robust model parameters.

Numerous methods have been proposed to attack or defend deep neural network models. Popular attack methods include fast gradient method (FGM) (Goodfellow et al., 2015), iterative fast gradient method (IFGM) (Kurakin et al., 2016), projected gradient descent (PGD) (Madry et al., 2017) and

---

[1]EPFL, Lausanne, Switzerland [2]Microsoft Research, Cambridge, UK. Correspondence to: Chen Liu <chen.liu@epfl.ch>.

CW attack (Carlini & Wagner, 2017). Most attack methods search for the adversarial example by utilizing the gradient of loss objective w.r.t. the input data. On the defenders' side, adversarial training (Szegedy et al., 2013), which augments the training data with adversarial examples, is a simple and popular method. It achieves the best empirical performance over other recent methods (Buckman et al., 2018; Ma et al., 2018; Guo et al., 2017; Dhillon et al., 2018; Xie et al., 2017; Song et al., 2017; Samangouei et al., 2018) under adversarial settings in Athalye et al. (2018).

However, recent work (Athalye et al., 2018) shows that these *uncertified* defense methods might fail when a stronger attack is applied. Therefore, finding a *provable* defense algorithm and *certifying* the level of robustness of an input data point have become active research topics. We need to guarantee that the models output consistent results in the worst cases under some conditions.

Following this line of works, we focus on the certification problem in this paper:

Given the label set $\mathcal{C}$, a classification model $f : \mathbb{R}^n \to \mathcal{C}$ and an input data point $\mathbf{x} \in \mathbb{R}^n$, we would like to find the largest neighborhood $\mathcal{S}$ around $\mathbf{x}$ such that $f(\mathbf{x}) = f(\mathbf{x}') \ \forall \mathbf{x}' \in \mathcal{S}$.

Many methods have been proposed for certifying a region bounded around a data point. Early studies use Satisfiability Modulo Theories (SMT) solvers (Huang et al., 2017b) or integer programming approaches (Lomuscio & Maganti, 2017). Despite some recent progress, these combinatorial methods still tend to suffer from superpolynomial time complexity (Katz et al., 2017; Wang et al., 2018) or requires solving a semi-definite programming problem (Raghunathan et al., 2018a;b).

More recently, Kolter & Wong (2017) and Wong et al. (2018) use a convex polytope relaxation to construct the bounds of the model's output logits. The gap between the logits of true and false labels is then minimized by primal-dual methods and can be trained by a dual network. Zhang et al. (2018) generalizes the method of Kolter & Wong (2017) to non-ReLU networks. Singh et al. (2018) further proposes a method of higher speed and precision based on the abstract interpretation of neural networks. These methods generally have *quadratic* complexity for strict bound.

All the studies mentioned above concentrate on certifying a *uniform* bound around a data point against adversarial perturbations. That is say, the certified regions $\mathcal{S}$ of all these algorithms are *uniform* across all input features. For example, when the input has two dimensions, the certified region is a square under l_∞ norm and a perfect round circle under l_2 norm.

In this paper, we explore the possibility of finding a *non-uniform* bounded data neighborhood without adversaries. In two-dimensional case above, the certified region would be a rectangle under l_∞ norm and an elliptical under l_2 norm. Such different level of robustness among input features has already found interest in the literature, such as Tsipras et al. (2018). Indeed, robust features should have larger perturbation tolerance than non-robust ones. Under *non-uniform* settings, we can have larger certified regions and a quantitative robustness metric for different input features.

Perhaps more importantly, we can use *non-uniform* bound as a tool to study the decision boundaries of different models. This is fundamental regarding understanding the robustness property of neural network models. Our work is an important step towards that.

We summarize the contributions of our work below:

- We provide a framwork to estimate the bounds of output logits in general feedforward neural network given non-uniform adversarial budget $\boldsymbol{\epsilon}$. Any existing bounding estimation algorithm can fit this framework as long as its bound is diferentiable w.r.t. $\boldsymbol{\epsilon}$

- In order to find the certified non-uniform bounds of the largest volumes, we formulate the goal as a constrained optimization problem, make relaxations and solve the relaxed problem by the augmented Langrangian method.

- Our method can find non-uniform bounds of larger volumes than uniform ones, revealing at least three benefits of robust models: certified non-uniform bounds of larger volumes, better interpretability and higher geometric similarity.

We formalize our non-uniform bound certification problem in Section 2 and propose solution algorithms in Section 3. We provide experimental evidence in Section 4, followed up by extensions and future works in Section 5. Conclusions can be found in Section 6.

2. Problem Formulation

We start with a N-layer fully connected neural network, parameterized by $\{\mathbf{W}^{(i)}, \mathbf{b}^{(i)}\}_{i=1}^{N-1}$:

$$\mathbf{z}^{(i+1)} = \mathbf{W}^{(i)}\hat{\mathbf{z}}^{(i)} + \mathbf{b}^{(i)} \quad i = 1, 2, ..., N-1$$
$$\hat{\mathbf{z}}^{(i)} = \sigma(\mathbf{z}^{(i)}) \qquad i = 2, 3, ..., N-1 \tag{1}$$

where $\{\mathbf{z}^{(i)}, \hat{\mathbf{z}}^{(i)}\}_{i=1}^{N-1}$ are pre- and post- activation values in each layer. The input data and the output logits of the neural network are $\hat{\mathbf{z}}^{(1)}$ and $\mathbf{z}^{(N)}$ respectively. σ is a nonlinear function which can be ReLU, sigmoid, tanh, etc. We use $n_1, n_2, ..., n_N$ to denote the number of neurons in each layer, so $\mathbf{W}^{(i)} \in \mathbb{R}^{n_{i+1} \times n_i}$ and $\mathbf{b}^{(i)} \in \mathbb{R}^{n_{i+1}}$

In this work, the adversarial budget for an input data point $\mathbf{x} \in \mathbb{R}^{n_1}$ is represented by a non-uniform bounded region $\mathcal{S}_{\boldsymbol{\epsilon}}^{(p)}(\mathbf{x})$, parameterized by $\boldsymbol{\epsilon} \in \mathbb{R}^{n_1}$. We define set $\mathcal{S}_{\boldsymbol{\epsilon}}^{(p)}(\mathbf{x})$ as $\{\hat{\mathbf{z}}^{(1)} = \mathbf{x} + \boldsymbol{\epsilon} \odot \mathbf{v} \|\mathbf{v}\|_p \le 1\}$ based on l_p norm. In most parts of this work, we focus on the l_∞ case and discuss the potential extension to other norms later. For simplicity, we use $\mathcal{S}_{\boldsymbol{\epsilon}}(\mathbf{x})$ to represent $\mathcal{S}_{\boldsymbol{\epsilon}}^{(\infty)}(\mathbf{x})$.

Now, we would like to find the certified region $\mathcal{S}_{\boldsymbol{\epsilon}}(\mathbf{x})$ of the largest volume, measured by $\Pi_{j=0}^{n_1-1}\epsilon_j$, in which the model outputs consistent label. Formally, for a data point labeled as the category $c \in \{0, 1, ..., n_N-1\}$, the problem we focus on is formulated below:

$$\min_{\boldsymbol{\epsilon}} \quad \left\{ -\sum_{j=0}^{n_1-1} \log \epsilon_j \right\}$$
$$\hat{\mathbf{z}}^{(1)} \in \mathcal{S}_{\boldsymbol{\epsilon}}(\mathbf{x})$$
$$\mathbf{z}^{(i+1)} = \mathbf{W}^{(i)}\hat{\mathbf{z}}^{(i)} + \mathbf{b}^{(i)} \qquad i = 1, 2, ..., N-1$$
$$\hat{\mathbf{z}}^{(i)} = \sigma(\mathbf{z}^{(i)}) \qquad i = 2, 3, ..., N-1$$
$$z_c^{(N)} - z_j^{(N)} \ge \delta \qquad j = 0, 1,..., n_N-1; j \ne c \tag{2}$$

As a common practice, we minimize the negative logarithm of $\Pi_{j=0}^{n_1-1}\epsilon_j$ because it is convex and more stable numerically. δ is a small positive constant to make sure the logits of the true label is strictly higher than others. We can see that if we constrain $\boldsymbol{\epsilon}$ by a scalar γ: $\boldsymbol{\epsilon} = \gamma \mathbf{1}$, the problem can be reduced to uniform bound certification problem, the one solved by Kolter & Wong (2017); Wong et al. (2018); Raghunathan et al. (2018a); Zhang et al. (2018); Singh et al. (2018).

Before going into the details of the algorithm, we also introduce the notation used in the sequel. We use $\mathbf{l}^{(i)}$ and $\mathbf{u}^{(i)}$ to represent the lower and upper bound of pre-activation values in the i-th layer respectively i.e. $\mathbf{l}^{(i)} \le \mathbf{z}^{(i)} \le \mathbf{u}^{(i)}$. For a tensor $\mathbf{T}$, $\mathbf{T}_-$ means all its negative elements i.e. $\mathbf{T}_- = \min(\mathbf{T}, 0)$. Similarity, we can define $\mathbf{T}_+$. $\odot$ is elementwise product operation between two tensors. Bracketed superscripts are used to index tensors while subscripts mean the elements in a tensor. Scalars are broadcast when

it is added or subtracted from tensors. The equality and inequality relations in this paper are all elementwise.

3. Algorithm

In this section, following the work of Kolter & Wong (2017); Singh et al. (2018); Zhang et al. (2018), we derive a linear approximation to bound the nonlinear activation function σ. This allows us to relax problem (2) into an optimization problem with bounds parameterized as a (nonlinear) function of ϵ. Then we show that we can compute the gradient of this bound with respect to ϵ efficiently and solve the relaxed problem using the augmented Lagrangian method.

3.1. Linear Approximation of Activation Functions

For an activation function $\sigma(x)$, which is nonlinear and monotonic, and x bounded by $l \leq x \leq u$, we can linearize it by two linear functions with the same slope: $kx + m_1 \leq \sigma(x) \leq kx + m_2$. k, m_1 and m_2 all depend on the bounds l and u. They are chosen in a way such that the gap $m_2 - m_1$ between bias terms shall be as small as possible.

For example, if σ is ReLU function: $\sigma(x) = \max(0, x)$, we have:

$$
k = \begin{cases} 0 & l \leq u \leq 0 \\ \dfrac{u}{u-l} & l < 0 < u \\ 1 & 0 \leq l \leq u \end{cases}, m_2 = \begin{cases} 0 & l \leq u \leq 0 \\ -\dfrac{ul}{u-l} & l < 0 < u \\ 0 & 0 \leq l \leq u \end{cases}
$$

$$
m_1 = 0
$$

(3)

Here, we find $m_1 = m_2$ when $l \leq u \leq 0$ or $0 \leq l \leq u$. Therefore, the linear approximation error arises for ReLU only when $l < 0 < u$.

For a general activation function $\sigma(x)$, we can consider functions $\tilde{m}_1(k) \leq \sigma(x) - kx$ and $\tilde{m}_2 \geq \sigma(x) - kx$ when $x \in [l, u]$. Then k is chosen by minimizing the margin $\tilde{m}_2 - \tilde{m}_1$. Zhang et al. (2018) extends Kolter & Wong (2017)'s methods in a similar way but they use numerical methods to obtain k. Here k needs to have analytical form, because we need the gradients of bounds w.r.t. ϵ.

Consider a vector $\mathbf{x}$ bounded by $\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$, we can bound $\sigma(\mathbf{x})$ by $\mathbf{D}\mathbf{x} + \mathbf{m}_1 \leq \sigma(\mathbf{x}) \leq \mathbf{D}\mathbf{x} + \mathbf{m}_2$ where $\mathbf{D}$ is a diagonal matrix and $\mathbf{m}_1, \mathbf{m}_2$ are bias vectors. Equivalently, we can say $\forall \mathbf{x} : \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \exists \mathbf{m} : \mathbf{m}_1 \leq \mathbf{m} \leq \mathbf{m}_2$ such that $\sigma(\mathbf{x}) = \mathbf{D}\mathbf{x} + \mathbf{m}$.

For a fully connected neural network defined in equation (1), we linearize $\hat{\mathbf{z}}^{(i)}$ by $\hat{\mathbf{z}}^{(i)} = \sigma(\mathbf{z}^{(i)}) = \mathbf{D}^{(i)}\mathbf{z}^{(i)} + \mathbf{m}^{(i)}$ under constraint $\mathbf{m}_1^{(i)} \leq \mathbf{m}^{(i)} \leq \mathbf{m}_2^{(i)}$ given the bound $\mathbf{l}^{(i)} \leq \mathbf{z}^{(i)} \leq \mathbf{u}^{(i)}$. For the input layer, the matrix $\mathbf{D}^{(1)}$ is

identity and the bias term is bounded by ϵ: $-\epsilon \leq \mathbf{m}^{(1)} \leq \epsilon$. If we unfold the linear approximations, the output of each layer $\mathbf{z}^{(i)}$ can be expressed in the following way:

$$
\begin{aligned}
\mathbf{z}^{(i)} &= \mathbf{W}^{(i-1)}(\sigma(\mathbf{W}^{(i-2)}(... \\
&\quad (\mathbf{W}^{(1)}(\mathbf{x} + \mathbf{m}^{(1)}) + \mathbf{b}^{(1)})...) + \mathbf{b}^{i-2})) + \mathbf{b}^{(i-1)} \\
&= \mathbf{W}^{(i-1)}(\mathbf{D}^{(i-1)}(\mathbf{W}^{(i-2)}(... \\
&\quad (\mathbf{W}^{(1)}(\mathbf{x} + \mathbf{m}^{(1)}) + \mathbf{b}^{(1)})...) + \mathbf{b}^{(i-2)}) + \mathbf{m}^{(i-1)}) + \mathbf{b}^{(i-1)} \\
&= \left(\Pi_{j=2}^{i-1}\mathbf{W}^{(j)}\mathbf{D}^{(j)} \right) \mathbf{W}^{(1)}\mathbf{x} + \sum_{h=1}^{i-1} \left(\Pi_{j=h+1}^{i-1}\mathbf{W}^{(j)}\mathbf{D}^{(j)} \right) \mathbf{b}^{(h)} \\
&\quad + \sum_{h=1}^{i-1} \left(\Pi_{j=h+1}^{i-1}\mathbf{W}^{(j)}\mathbf{D}^{(j)} \right) \mathbf{W}^{(h)}\mathbf{m}^{(h)}
\end{aligned}
$$

(4)

3.2. Lower and Upper Bound Estimation

According to equation (4), for the output layer or any intermediate layer $\mathbf{z}^{(i)}$ of a model, the only variables on the right hand side are $\{\mathbf{m}^{(h)}\}_{h=1}^{i-1}$. The bound of $\mathbf{z}^{(i)}$ can be obtained immediately from the bounds of $\{\mathbf{m}^{(h)}\}_{h=1}^{i-1}$. Once we have the bound of $\mathbf{z}^{(i)}$, we can then obtain $\mathbf{D}^{(i)}$, $\mathbf{m}_1^{(i)}$ and $\mathbf{m}_2^{(i)}$ to bound $\mathbf{z}^{(i+1)}$. Such process can be done iteratively until we obtain the lower bound $\mathbf{l}^{(N)}$ and upper bound $\mathbf{u}^{(N)}$ of the output logits $\mathbf{z}^{(N)}$. The computational complexity of FLOP operations in this algorithm is $O(N^2 n^3)$ where n is the maximum number of neurons in a hidden layer. We call this algorithm *quadratic algorithm* and provide pseudo code in **Algorithm** 3 in Appendix A. This algorithm is the same as CROWN in Zhang et al. (2018) except that we use the same slope for upper and lower bound of the nonlinearity. **Algorithm** 3 only maintains one list of matrices i.e. $\mathbf{M}^{(j)}$ in each iteration while CROWN needs 4 such matrices and thus consumes much more memory.

Theoretically, any convex hull of the nonlinear function σ in $[l, u]$ leads to a valid bound of the output logits (Salman et al., 2019), therefore, the smallest convex hull corresponds to the tightest bound. However, in practice we can hardly find an analytical form of the smallest convex hull for arbitrary σ. It is a trade-off between the tightness and the complexity. In this work, we use two parallel line to bound the nonlinear function within a given range, so our convex hull here is a parallelogram.

On the other hand, we can obtain a naive layerwise bound by iteratively calculating the bound of a layer based on the bound of the immediate previous layer. Compared with *quadratic algorithm*, the complexity of FLOP operations in this algorithm is $O(Nn^2)$ [1] and we call it *simple algorithm* The pseudo code is in **Algorithm** 4 in Appendix A. Because

---

[1] In *simple algorithm*, there are only matrix-vector multiplications, whose complexity are $O(n^2)$ each.

of its efficiency, *Simple algorithm* has been incorporated into the algorithms in Gowal et al. (2018) to train robust models.

Although Kolter & Wong (2017) has empirically showed that *quadratic algorithm* typically obtains better bounds than *simple algorithm*, we find that both algorithms are actually complementary. On one hand, *quadratic algorithm* linearizes the activation functions and the difference between upper and lower bounds come from the flexibility of $\{\mathbf{m}^{(h)}\}_{h=1}^{i-1}$, which grows moderately with the number of layers. *Simple algorithm* calculates the bounds of each layer iteratively and the error may propagate much faster. However, *simple algorithm* calculates the activation function exactly without any approximation. Therefore, neither algorithm is *guaranteed* to be better than the other under all circumstances. Roughly speaking, *simple algorithm* works better for networks of few layers ('shallow' networks) or networks containing layers of very few neurons ('thin' networks), while *quadratic algorithm* is better in other cases. Detailed discussion and examples for comparison are deferred in Appendix B.

In this work, we incorporate *simple algorithm* into *quadratic algorithm* for a better bound in each layer. The pseudo code is given as **Algorithm** 1 below. The maximum and minimum operators in line 15 and 16 are applied elementwisely.

Algorithm 1 Bound Estimation

1: Input: Parameters $\{\mathbf{W}^{(i)}, \mathbf{b}^{(i)}\}_{i=1}^{N-1}$, perturbation set $\mathcal{S}_{\boldsymbol{\epsilon}}(\mathbf{x})$.
2: $\mathbf{l}^{(2)} = \mathbf{W}^{(1)}\mathbf{x} - \mathbf{W}_+^{(1)}\boldsymbol{\epsilon} + \mathbf{W}_-^{(1)}\boldsymbol{\epsilon} + \mathbf{b}^{(1)}$
3: $\mathbf{u}^{(2)} = \mathbf{W}^{(1)}\mathbf{x} - \mathbf{W}_-^{(1)}\boldsymbol{\epsilon} + \mathbf{W}_+^{(1)}\boldsymbol{\epsilon} + \mathbf{b}^{(1)}$
4: $\mathbf{M}^{(1)} = \mathbf{W}^{(1)}$
5: $\phi^{(2)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$
6: **for** i = 2, ..., N − 1 **do**
7:     Calculate $\mathbf{D}^{(i)}, \mathbf{m}_1^{(i)}, \mathbf{m}_2^{(i)}$ based on $\mathbf{l}^{(i)}$ and $\mathbf{u}^{(i)}$
8:     $\mathbf{l}_{simp}^{(i+1)} = \mathbf{W}_+^{(i)}\sigma(\mathbf{l}^{(i)}) + \mathbf{W}_-^{(i)}\sigma(\mathbf{u}^{(i)})$
9:     $\mathbf{u}_{simp}^{(i+1)} = \mathbf{W}_-^{(i)}\sigma(\mathbf{l}^{(i)}) + \mathbf{W}_+^{(i)}\sigma(\mathbf{u}^{(i)})$
10:    $\mathbf{M}^{(j)} = \mathbf{W}^{(i)}\mathbf{D}^{(i)}\mathbf{M}^{(j)}$ for $j = 1, ..., i - 1$
11:    $\mathbf{M}^{(i)} = \mathbf{W}^{(i)}$
12:    $\phi^{(i+1)} = \mathbf{W}^{(i)}\mathbf{D}^{(i)}\phi^{(i)} + \mathbf{b}^{(i)}$
13:    $\mathbf{l}_{quad}^{(i+1)} = \phi^{(i+1)} + \sum_{j=1}^i \left(\mathbf{M}_-^{(j)}\mathbf{m}_2^{(j)} + \mathbf{M}_+^{(j)}\mathbf{m}_1^{(j)} \right)$
14:    $\mathbf{u}_{quad}^{(i+1)} = \phi^{(i+1)} + \sum_{j=1}^i \left(\mathbf{M}_-^{(j)}\mathbf{m}_1^{(j)} + \mathbf{M}_+^{(j)}\mathbf{m}_2^{(j)} \right)$
15:    $\mathbf{l}^{(i+1)} = \max(\mathbf{l}_{simp}^{(i+1)}, \mathbf{l}_{quad}^{(i+1)})$
16:    $\mathbf{u}^{(i+1)} = \min(\mathbf{u}_{simp}^{(i+1)}, \mathbf{u}_{quad}^{(i+1)})$
17: **end for**
18: Output: Bounds $\{\mathbf{l}^{(i)}, \mathbf{u}^{(i)}\}_{i=2}^N$

3.3. Gradient of Perturbation Budget ϵ

Algorithm 1 provides the algorithm to estimate the bound of output logits for any input perturbation $\mathcal{S}_{\boldsymbol{\epsilon}}(\mathbf{x})$. In uniform bound certification problem, we constrain $\boldsymbol{\epsilon} = \gamma\mathbf{1}$ and have one dimensional variable $\gamma \in \mathbb{R}$ to optimize. In this

case, the optimality can be found by binary search or line search (Zhang et al., 2018). However, in non-uniform bound certification problem, we have n_1 dimensional variable $\boldsymbol{\epsilon}$ to optimize. It becomes necessary to estimate a good direction to update $\boldsymbol{\epsilon}$. This is why we consider gradient methods, and fortunately the (sub)gradient $\frac{\partial \mathbf{l}^{(N)}}{\partial \boldsymbol{\epsilon}}$ and $\frac{\partial \mathbf{u}^{(N)}}{\partial \boldsymbol{\epsilon}}$ can obtained according to **Algorithm** 1.

Based on the linear approximation as discussed in Section 3.1, standard back-propagation can be applied to calculate the (sub)gradients of the final bounds w.r.t. $\boldsymbol{\epsilon}$. Alternatively, similar to **Algorithm** 1, the gradient can be calculated recursively using chain rule. Unlike back-propagation, this method does not need to wait for the termination of **Algorithm** 1 and can be calculated on the fly, which is beneficial in the distributed settings. The pseudo code is provided in Appendix A.

3.4. Optimization by the Augmented Lagrangian Method

Now, we have done all the preparation to solve the relaxation of problem (2). By bound estimation, we rewrite the problem below:

$$\min_{\boldsymbol{\epsilon}, \mathbf{y} \geq 0} \left\{ -\sum_{j=0}^{n_1-1} \log \epsilon_j \right\} \tag{5}$$
$$s.t.\ l_c^{(N)} - \mathbf{u}_{j \neq c}^{(N)} - \delta = \mathbf{y}$$

Here $\mathbf{l}^{(N)}$ and $\mathbf{u}^{(N)}$ are functions of $\boldsymbol{\epsilon}$ given by **Algorithm** 1. $\mathbf{u}_{j \neq c}^{(N)} \in \mathbb{R}^{n_N-1}$ is the concatenation of all output logits except true label c. $\mathbf{y}(\geq 0) \in \mathbb{R}^{n_N-1}$ is a slack variable that ensures the nonnegativity of the term on the left hand side. For simplicity, we define $\mathbf{v} := l_c^{(N)} - \mathbf{u}_{j \neq c}^{(N)} - \delta$ as a function of $\boldsymbol{\epsilon}$.

Note that we have replaced constraint $\mathbf{z}_c^{(N)} - \mathbf{z}_j^{(N)} \geq \delta$ in problem (2) by a stronger version $l_c^{(N)} - \mathbf{u}_{j \neq c}^{(N)} \geq \delta$. Therefore, the optimality of problem (5) provides the upper bound of the original minimization problem (2).

We can further rewrite the problem (5) into a min-max problem using augmented Lagragian method (Hestenes, 1969; Powell, 1969) by introducing the dual variable $\boldsymbol{\lambda} \in \mathbb{R}^{n_N-1}$ and the coefficient $\rho \in \mathbb{R}^+$. The dual problem to solve is below:

$$\max_{\boldsymbol{\lambda}} \min_{\boldsymbol{\epsilon}, \mathbf{y} \geq 0} - \left(\sum_{j=0}^{n_1-1} \log \epsilon_j \right) + \langle \boldsymbol{\lambda}, \mathbf{v} - \mathbf{y} \rangle + \frac{\rho}{2} \|\mathbf{v} - \mathbf{y}\|_2^2 \tag{6}$$

The inner minimization problem is a quadratic form of $\mathbf{y}$, so

the optimal $\mathbf{y}$ has the analytical solution: $\mathbf{y} = \max(0, \mathbf{v} + \frac{1}{\rho}\boldsymbol{\lambda})$. Plug the solution in the problem and we can optimize $\boldsymbol{\epsilon}$ by gradient descent. The pseudo code is in **Algorithm** 2.

Algorithm 2 Optimization for $\boldsymbol{\epsilon}$

1: Input: Parameters $\{\mathbf{W}^{(i)}, \mathbf{b}^{(i)}\}_{i=1}^{N-1}$, original bounds $\boldsymbol{\epsilon}_0$, iterations M, augmented coefficient $\{\rho^{(i)}\}_{i=1}^{M}$, decaying factor η.
2: $\boldsymbol{\epsilon} = \boldsymbol{\epsilon}_0, \boldsymbol{\lambda} = 0$
3: **for** $i = 1, 2, ..., M$ **do**
4:     Update $\boldsymbol{\epsilon}$ by minimizing (6) with optimal $\mathbf{y}$ plugged in.
5:     $\boldsymbol{\lambda} = \boldsymbol{\lambda} + \rho^{(i)}(\mathbf{v} - \mathbf{y})$
6: **end for**
7: **while** $\mathbf{v} \geq 0$ is not satisfied **do**
8:     $\boldsymbol{\epsilon} = \eta\boldsymbol{\epsilon}$
9: **end while**
10: Output: $\boldsymbol{\epsilon}$

Similar to penalty method, the coefficient $\{\rho^{(i)}\}_{i=1}^{M}$ in **Algorithm** 2 is a non-decreasing sequence to enforce the constraint. However, the Lagrange multiplier term makes it unnecessary to increase $\rho^{(i)}$ to $+\infty$. Actually, $\rho^{(i)}$ can stay much smaller here to solve the problem, which avoids numerical instability caused by ill-conditioning.

The minimization in line 4 is solved by gradient methods. In practice, gradient explosion might happen when $\boldsymbol{\epsilon}$ is small or $\rho^{(i)}$ is big. To avoid overshooting, we apply gradient rescaling to constrain the l_2 norm of the gradient. The term $\log \epsilon_j$ implicitly constrains ϵ_j to be positive, so we reparametrize $\epsilon_j = \zeta_j^2$ and optimize vector $\boldsymbol{\zeta}$.

The last while-loop in line 7 is to ensure the output $\boldsymbol{\epsilon}$ meets the hard constraints. The decaying factor η is close to 1 and is set 0.99 in practice. When $\rho^{(i)}$ is large, the while-loop would break after very few iterations.

4. Experiments

In this Section, we compare our certified non-uniform bounds with uniform bounds. We also use our algorithm as a tool to explore the decision boundaries of different models. All the experiments here are implemented in the framework of PyTorch and can be finished within several hours on a single NVIDIA Tesla GPU machine.

Because any algorithm of estimating the output logits can be incorporated into our framework of computing non-uniform bounds, our main focus in this section is the comparison between uniform and non-uniform bounds based on the same estimation algorithm of output logits.

4.1. Synthetic Data

We first validate our algorithm using 2-D synthetic data so that we can visualize the certified bounds.

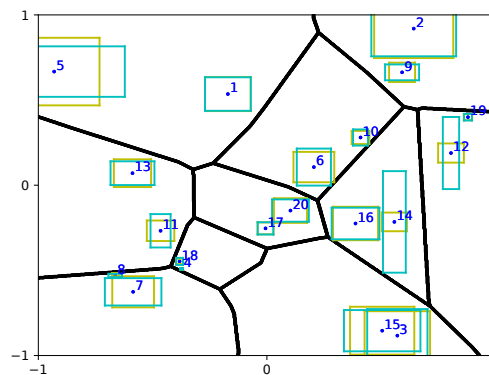We generate 10 random 2D data points in the space of

Figure 1. A simple example on the synthetic data. 20 points with their certified uniform (yellow) and non-uniform (blue) bounds are shown above. The real decision boundary is shown as black lines.

$[-1, 1]^2$ labeled $\{0, 1, ..., 9\}$ as seeds. Another 10000 random points in $[-1, 1]^2$ are then generated and assigned the same label as the closest seeds. 90% of the data points are in the training set and the rest are reserved for testing.

The model here is a ReLU fully-connected neural network with two hidden layers of 10 neurons. Since the boundary between different categories are piecewise linear in this case, the model is shown to have enough capacity and achieve an accuracy of more than 99.9% in the test set.

Figure 1 demonstrates the results of uniform (yellow) and non-uniform (blue) bounds with two random points in each category. We can clearly see the bounds calculated by our algorithm are reasonably tight and areas covered by non-uniform bounds are larger than those of uniform bounds. Although the larger volumes do not necessarily mean the bounds are larger for both features, bounds of some feature are extended in compensation for the other. We can say the features are more robust when their bounds are larger.

In some cases, the non-uniform bounds can be significantly larger than uniform bounds (e.g. point 12, 14 in Figure 1). This typically means considerable difference in robustness between two input features.

4.2. Real Datasets

In this part, we run our algorithm on real datasets, including MNIST, Fashion-MNIST and SVHN (Netzer et al., 2011). All of them are popular benchmarks for image classification and contain tens of thousand images. MNIST and Fashion-MNIST are 28×28 gray-scale images, while SVHN are 32×32 colored images. Unless specified, all pixel values of images are normalized in the range of $[-1, 1]$.

Besides a metric of feature robustness, the non-uniform

Dataset	Architecture	Adversary	Accuracy (%)	Uniform Bound	Non-uniform Bound	Ratio
MNIST	100-100-100	-	99.2	0.0295	0.0349	1.183
		PGD, $\tau = 0.1$	98.1	0.0692	0.1678	2.425
	300-300-300	-	98.0	0.0309	0.0350	1.133
		PGD, $\tau = 0.1$	98.9	0.0507	0.1404	2.769
	500-500-500	-	98.5	0.0319	0.0360	1.129
		PGD, $\tau = 0.1$	98.8	0.0436	0.1167	2.677
Fashion MNIST	1024-1024-1024	-	98.7	0.0397	0.0518	1.305
		PGD, $\tau = 0.1$	99.0	0.0446	0.1134	2.543
SVHN	1024-1024-1024	-	84.3	0.0022	0.0072	3.273
		PGD, $\tau = 0.1$	78.2	0.0054	0.0281	5.204

Table 1. Average of uniform and non-uniform bounds in the test sets. The architecture column means the number of neurons in hidden layers. The accuracy column means the value of clean accuracy. The ratio is the values of non-uniform bounds over uniform bounds.

bounds also can be used as a tool to explore the decision boundaries of models. In the following subsections, we investigate the difference between the decision boundaries of robust and non-robust models.

4.2.1. ROBUSTNESS AND VOLUME OF BOUNDS

As Table 1 shows, we train different models for different datasets in different ways. To obtain robust models, we do adversarial training based on PGD (Madry et al., 2017) attacks. To the best of our knowledge, this is the way to obtain the most robust model empirically studied in Athalye et al. (2018). We set the perturbation budget τ of PGD to be 0.1 and search for adversarial examples for 20 iterations. We call models adversarially trained by PGD *robust models* to distinguish from *normal models*.

We report results based on data in the test sets here. To make the results of non-uniform bound comparable with uniform bound, we take the geometric average values $\left(\Pi_{j=0}^{n_1-1} \epsilon_j\right)^{\frac{1}{n_1}}$. Table 1 shows the average bounds in different settings, it is clear that non-uniform bounds consistently certify areas of larger volumes.

We notice that the ratio of non-uniform bound over uniform bound is significantly larger in the cases of robust models. Figure 2 shows the histogram of bound per feature for normal and robust models on a randomly picked image in MNIST. Compared with the normal model, the bounds of some features for the robust model can be as large as 0.4, much larger than the value of τ. This observation means the decision boundary of the robust model is almost aligned in some dimensions corresponding to some features, which makes it possible for our algorithm to extend the bounds of those features without sacrificing the bounds of the others much. It also implies robust models tend to drop irrelevant features and rely on fewer features when making predictions. We put more results from other MNIST models as well as models on Fashion-MNIST and SVHN dataset to support our claim in Appendix C.1.
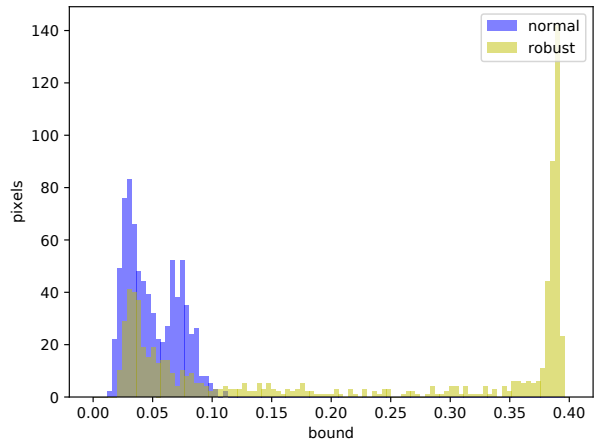

Figure 2. The distributions of bounds per feature for normal and robust models in a randomly picked image. Both models have three hidden layers of 300 neurons. The uniform bounds for normal and robust models are 0.0402 and 0.0568 respectively, while the corresponding geometric average values of non-uniform bounds are 0.0450 and 0.1462 respectively.

4.2.2. ROBUSTNESS AND MODEL INTERPRETABILITY

Given an image and a neural network model, our algorithm can obtain a non-uniform bound parameterized by $\epsilon \in \mathbb{R}^{n_1}$. For high dimensional images, we can not plot the rectangular bounds. However, we can visualize ϵ just like images. We call them *bounding maps* and use the same rescaling factor [2] to visualize them in this paper.

To study the property of bounding maps, we take a simple example of binary classification: to distinguish digit '1' from '7' in MNIST dataset. We use model with three

---

[2]All bounding map figures in this paper are plot based on $\mathbf{1} - 5\epsilon$, so darker pixels in bounding maps mean larger bounds.

	Mean Cosine	Minimum Cosine
Normal Model	0.9774	0.5038
Robust Model	0.9964	0.9104

Table 2. The mean and minimum of cosine similarity between all pairs of ϵ in the test set. We use MNIST classification model with 300 neurons in each hidden layer.
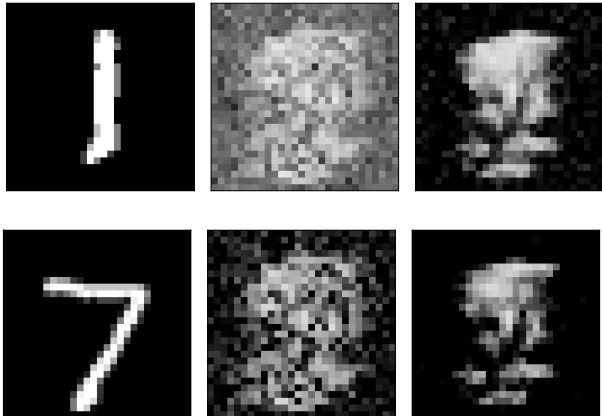
Figure 3. An example of bounding maps of images of digit '1' and '7'. (Left) The original images, (Middle) Bounding map of a normal model. (Right) Bounding map of a robust model.

hidden layers of 100 neurons and train it both in normal and adversarial way. We visualize the bounding maps of both models for two example images in Figure 3.

For the normal model, the bounding maps are noisy and can hardly reveal the patterns of the input data. However, for the robust model, the bounding maps can capture some intrinsic characteristics of the input data. In the case of digit '1' and '7', people typically distinguish them by the horizontal stroke which digit '7' has and '1' does not. This corresponds to the relatively smaller bounds of features in the middle above of the images. It indicates the decision boundaries are closer to the data points in the directions of these features and the model puts more weight on these features to make predictions. On the contrary, both digit '1' and '7' have a vertical stroke, we can correspondingly see a dark clear vertical stroke in the bounding maps of the robust model. Such phenomenon can be reproduced in many other examples, more are available in Appendix C.2.

We need to mention similar property of robust models is found in Tsipras et al. (2018) but from a more microscopic perspective. Tsipras et al. (2018) visualizes the gradients of the model's loss function w.r.t the input data and finds that the gradients for robust models are *significantly more interpretable*, while the gradients for normal models are generally noise. Our investigations are more on a macroscopic level, our non-uniform bounds explore the shape of model's decision boundary but we have the same claim: *robust models are more interpretable*.

4.2.3. ROBUSTNESS AND DECISION BOUNDARY

In Figure 3, similar patterns of bounding maps are found for the same model but different input images. Thus, we calculate the cosine similarity of ϵ for two images, since the

direction of ϵ indicates the shape of non-uniform bounds. For examples of normal and robust models, we report the average and minimum values in all image pairs of the test set in Table 2. The full results are available in Table 3 in Appendix C.3. It is clear that the values of ϵ for different images but the same model are highly correlated, which indicates the geometric similarity of non-uniform bounds. [3] What's more, such correlation is even stronger in the cases of robust models.

The high correlation means some features are consistently more robust than the other features in different input data points. Since most ϵ are almost collinear, the direction of them can be regarded as a quantitative and data-agnostic metric measuring the robustness of input features. It is also beneficial to use this direction as a prior when we estimate the non-uniform bound for a new data point.

Since the shape of the non-uniform bound reveals the decision boundary, high correlation of ϵ also indicates the uniformity of the direction of the decision boundary. Formally, in a n_1 dimensional input space, there exists a subspace $\mathcal{X}$ of dimensionality $n_1' \ll n_1$ that contains most directions of decision boundary around the data manifold. This is consistent with what Moosavi-Dezfooli et al. (2017) points out.

An extreme example is the classifier whose decision boundary is linear, the non-uniform bound of the largest volume for any input data has exactly the same shape. That is to say, the values of ϵ for any input data point are exactly collinear and $n_1' = 1$ in this case. Our experimental results show the stronger correlation of ϵ in the cases of robust models. This implies the most directions of a robust model's decision boundary can be obtained in a subspace of even lower dimensionality than a normal model. The decision boundary of a robust model should be simpler in some sense.

5. Discussion

In this section, two straightforward extensions are shown to make our algorithms adapt to other settings. We also discuss the potential future works to polish the algorithms.

---

[3]For two random vectors uniformly distributed in $[0, 1]^{784}$, the expectation of cosine similarity between them is around 0.75. The expectation decreases for random vectors in higher dimensions.

5.1. Extensions

5.1.1. OTHER NETWORK ARCHITECTURES

Some previous works have some assumptions on the architecture of the neural network. For example, Kolter & Wong (2017) and Weng et al. (2018) assume ReLU network; Raghunathan et al. (2018a) only works for network with only one hidden layer. Wong et al. (2018) and Zhang et al. (2018) generalize the method of Kolter & Wong (2017) and Weng et al. (2018) respectively to general feedforward networks.

Although our previous analysis in Section 3 is based on the fully-connected network, our framework is modularized and can generalize naturally to general feedforward networks in the same way as Wong et al. (2018). Our method is from the primal perspective while Wong et al. (2018) focuses on the dual problem.

For example, the convolutional layers can be reparameterized as feedforward layers of sparse weight matrices and shared variables. The max-pooling layers can be considered as non-linear functions and be linearized by methods in Section 3.1. Our framework can also be applied to network with shortcut connections, including popular Residual Network (ResNet) (He et al., 2016) and Densely Connected Network (DenseNet) (Huang et al., 2017a). More details about this are available in Appendix D.

5.1.2. OTHER NORMS

Much attention of previous works is focused on bounds based on l_∞ norm (Zhang et al., 2018; Singh et al., 2018; Kolter & Wong, 2017), although some of them such as Kolter & Wong (2017) can be easily extended to attacks based on other norms. We claim that our methods can be extended to other norms in the same way. However, we also point out both previous works and this work implicitly favor l_∞ norm when estimating the bounds of output logits. This is because the neurons in intermediate layers are bounded in an *elementwise* manner, the bound for a specific neuron does not depend on the bound of any other neuron. This bound would be loose if the output $\mathbf{z}^{(i)}$, as a vector, can be better bounded by a norm other than l_∞ norm. It would be interesting to consider the bound of neurons in one layer jointly and derive a tighter bound for non-l_∞ norms.

5.2. Potential Future Works

Solving problem (2) exactly is difficult, because the exact range of output logits $\mathbf{z}^{(N)}$ when $\hat{\mathbf{z}}^{(1)} \in \mathcal{S}_\epsilon(\mathbf{x})$ is generally intractable. Therefore, we introduce a tractable bound of $\mathbf{z}^{(N)}$ and build our algorithms based on that. From the geometric perspective, the bound of $\mathbf{z}^{(N)}$ given by **Algorithm 1** provides a tractable envelope of the intractable decision boundary of the neural network. The algorithm to bound $\mathbf{z}^{(N)}$ is important, because an algorithm of a tighter bound reveals the decision boundary better and then leads to certified regions of larger volumes.

One possible direction to explore is to design faster and better algorithms than **Algorithm** 1, which still has looser bounds for larger ϵ or deeper networks. In addition, as **Algorithm** 1 is called in every iteration when we optimize ϵ in **Algorithm** 2, the complexity of the algorithm is also an issue. All algorithms discussed in this paper calculate the 'strict bounds' i.e. $\mathbf{l}^{(N)} \leq \mathbf{z}^{(N)} \leq \mathbf{u}^{(N)}$, it would be beneficial to design an algorithm for 'soft bounds' i.e. $\tilde{\mathbf{l}}^{(N)} \lesssim \mathbf{z}^{(N)} \lesssim \tilde{\mathbf{u}}^{(N)}$ but with much faster speed. We run 'soft bounds' algorithm first to accelerate the optimization and run 'strict bounds' algorithm at last to guarantee the hard constraints are satisfied.

Another direction towards certified regions of larger volumes and studying the decision boundary is to consider *oblique bounds* instead of *standard bounds*. The only difference is to introduce an additional orthogonal matrix $\mathbf{A}$ to parameterize the adversarial budget $\mathcal{S}_{\mathbf{A},\epsilon}^{(p)}(\mathbf{x}) := \{\hat{\mathbf{z}}^{(1)} = \mathbf{x} + \mathbf{A}(\epsilon \odot \mathbf{v}) | \|\mathbf{v}\|_p = 1\}$. More importantly, the elements in matrix $\mathbf{A}$ represents the correlation between different input features, which will give us more information about the shape of the decision boundary.

6. Conclusion

In this paper, we study the certified non-uniform bounds around input data points. We propose a general framework to estimate the output logits of different neural networks. The goal of finding the bounds of the largest volumes is then formulated as a constrained optimization problem and we solve it by the augmented Langragian method. Our experiments on synthetic data and real data show the non-uniform bounds have the larger volumes than uniform bounds. In addition, we use our algorithm as a tool to explore the decision boundaries of different models. Our results demonstrate at least three advantages of robust models: 1) the model relies on fewer features and has much larger certified non-uniform bounds; 2) the non-uniform bounds are significantly more interpretable; 3) the stronger geometric similarity of the non-uniform bounds gives a quantitative, data-agnostic metric of input features' robustness and implies a simpler decision boundary.

Acknowledgement

References

Athalye, A., Carlini, N., and Wagner, D. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

Buckman, J., Roy, A., Raffel, C., and Goodfellow, I. Thermometer encoding: One hot way to resist adversarial examples. 2018.

Carlini, N. and Wagner, D. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57. IEEE, 2017.

Dhillon, G. S., Azizzadenesheli, K., Lipton, Z. C., Bernstein, J., Kossaifi, J., Khanna, A., and Anandkumar, A. Stochastic activation pruning for robust adversarial defense. *arXiv preprint arXiv:1803.01442*, 2018.

Goodfellow, I., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2015.

Gowal, S., Dvijotham, K., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Mann, T., and Kohli, P. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

Guo, C., Rana, M., Cisse, M., and van der Maaten, L. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hestenes, M. R. Multiplier and gradient methods. *Journal of optimization theory and applications*, 4(5):303–320, 1969.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *CVPR*, volume 1, pp. 3, 2017a.

Huang, X., Kwiatkowska, M., Wang, S., and Wu, M. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pp. 3–29. Springer, 2017b.

Katz, G., Barrett, C., Dill, D. L., Julian, K., and Kochenderfer, M. J. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pp. 97–117. Springer, 2017.

Kolter, J. Z. and Wong, E. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 1(2):3, 2017.

Kurakin, A., Goodfellow, I., and Bengio, S. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

Lomuscio, A. and Maganti, L. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351*, 2017.

Ma, X., Li, B., Wang, Y., Erfani, S. M., Wijewickrema, S., Schoenebeck, G., Song, D., Houle, M. E., and Bailey, J. Characterizing adversarial subspaces using local intrinsic dimensionality. *arXiv preprint arXiv:1801.02613*, 2018.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Moosavi-Dezfooli, S.-M., Fawzi, A., Fawzi, O., and Frossard, P. Universal adversarial perturbations. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 86–94. Ieee, 2017.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 5, 2011.

Powell, M. J. A method for nonlinear constraints in minimization problems. *Optimization*, pp. 283–298, 1969.

Raghunathan, A., Steinhardt, J., and Liang, P. Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*, 2018a.

Raghunathan, A., Steinhardt, J., and Liang, P. Semidefinite relaxations for certifying robustness to adversarial examples. *arXiv preprint arXiv:1811.01057*, 2018b.

Salman, H., Yang, G., Zhang, H., Hsieh, C.-J., and Zhang, P. A convex relaxation barrier to tight robust verification of neural networks. *arXiv preprint arXiv:1902.08722*, 2019.

Samangouei, P., Kabkab, M., and Chellappa, R. Defensegan: Protecting classifiers against adversarial attacks using generative models. *arXiv preprint arXiv:1805.06605*, 2018.

Singh, G., Gehr, T., Mirman, M., Püschel, M., and Vechev, M. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*, pp. 10824–10835, 2018.

Song, Y., Kim, T., Nowozin, S., Ermon, S., and Kushman, N. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. *arXiv preprint arXiv:1710.10766*, 2017.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., and Madry, A. There is no free lunch in adversarial robustness (but there are unexpected benefits). *arXiv preprint arXiv:1805.12152*, 2018.

Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*, pp. 6367–6377, 2018.

Weng, T.-W., Zhang, H., Chen, H., Song, Z., Hsieh, C.-J., Boning, D., Dhillon, I. S., and Daniel, L. Towards fast computation of certified robustness for relu networks. *arXiv preprint arXiv:1804.09699*, 2018.

Wong, E., Schmidt, F., Metzen, J. H., and Kolter, J. Z. Scaling provable adversarial defenses. *arXiv preprint arXiv:1805.12514*, 2018.

Xie, C., Wang, J., Zhang, Z., Ren, Z., and Yuille, A. Mitigating adversarial effects through randomization. *arXiv preprint arXiv:1711.01991*, 2017.

Zhang, H., Weng, T.-W., Chen, P.-Y., Hsieh, C.-J., and Daniel, L. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems*, pp. 4939–4948, 2018.

A. Missing Algorithms

A.1. Bound Estimation Algorithms

Here we list the detailed pseudo code of *quadratic* and *simple* algorithms mentioned in Section 3.2. The complexity in matrix multiplications for **Algorithm** 3 and 4 and are $O(N^2)$ and $O(N)$ respectively.

Algorithm 3 *Quadratic* Bound Estimation

1: Input: Parameters $\{\mathbf{W}^{(i)}, \mathbf{b}^{(i)}\}_{i=1}^{N-1}$, perturbation set $\mathcal{S}_\epsilon(\mathbf{x})$.
2: $\mathbf{l}^{(2)} = \mathbf{W}^{(1)}\mathbf{x} - \mathbf{W}_+^{(1)}\boldsymbol{\epsilon} + \mathbf{W}_-^{(1)}\boldsymbol{\epsilon} + \mathbf{b}^{(1)}$
3: $\mathbf{u}^{(2)} = \mathbf{W}^{(1)}\mathbf{x} - \mathbf{W}_-^{(1)}\boldsymbol{\epsilon} + \mathbf{W}_+^{(1)}\boldsymbol{\epsilon} + \mathbf{b}^{(1)}$
4: $\mathbf{M}^{(1)} = \mathbf{W}^{(1)}$
5: $\phi^{(2)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$
6: **for** i = 2, ..., $N - 1$ **do**
7:    Calculate $\mathbf{D}^{(i)}, \mathbf{m}_1^{(i)}, \mathbf{m}_2^{(i)}$ based on $\mathbf{l}^{(i)}$ and $\mathbf{u}^{(i)}$
8:    $\mathbf{M}^{(j)} = \mathbf{W}^{(i)}\mathbf{D}^{(i)}\mathbf{M}^{(j)}$ for $j = 1, ..., i - 1$
9:    $\mathbf{M}^{(i)} = \mathbf{W}^{(i)}$
10:   $\phi^{(i+1)} = \mathbf{W}^{(i)}\mathbf{D}^{(i)}\phi^{(i)} + \mathbf{b}^{(i)}$
11:   $\mathbf{l}^{(i+1)} = \phi^{(i+1)} + \sum_{j=1}^{i} \left(\mathbf{M}_-^{(j)}\mathbf{m}_2^{(j)} + \mathbf{M}_+^{(j)}\mathbf{m}_1^{(j)}\right)$
12:   $\mathbf{u}^{(i+1)} = \phi^{(i+1)} + \sum_{j=1}^{i} \left(\mathbf{M}_-^{(j)}\mathbf{m}_1^{(j)} + \mathbf{M}_+^{(j)}\mathbf{m}_2^{(j)}\right)$
13: **end for**
14: Output: Bounds $\{\mathbf{l}^{(i)}, \mathbf{u}^{(i)}\}_{i=2}^{N}$

Algorithm 4 *Simple* Bound Estimation

1: Input: Parameters $\{\mathbf{W}^{(i)}, \mathbf{b}^{(i)}\}_{i=1}^{N-1}$, perturbation set $\mathcal{S}_\epsilon(\mathbf{x})$.
2: $\mathbf{l}^{(2)} = \mathbf{W}^{(1)}\mathbf{x} - \mathbf{W}_+^{(1)}\boldsymbol{\epsilon} + \mathbf{W}_-^{(1)}\boldsymbol{\epsilon} + \mathbf{b}^{(1)}$
3: $\mathbf{u}^{(2)} = \mathbf{W}^{(1)}\mathbf{x} - \mathbf{W}_-^{(1)}\boldsymbol{\epsilon} + \mathbf{W}_+^{(1)}\boldsymbol{\epsilon} + \mathbf{b}^{(1)}$
4: **for** i = 2, ..., $N - 1$ **do**
5:    $\mathbf{l}^{(i)} = \sigma(\mathbf{l}^{(i)})$
6:    $\mathbf{u}^{(i)} = \sigma(\mathbf{u}^{(i)})$
7:    $\mathbf{l}^{(i+1)} = \mathbf{W}_+^{(i+1)}\mathbf{l}^{(i)} + \mathbf{W}_-^{(i+1)}\mathbf{u}^{(i)} + \mathbf{b}^{(i)}$
8:    $\mathbf{u}^{(i+1)} = \mathbf{W}_-^{(i+1)}\mathbf{l}^{(i)} + \mathbf{W}_+^{(i+1)}\mathbf{u}^{(i)} + \mathbf{b}^{(i)}$
9: **end for**
10: Output: Bound $\{\mathbf{l}^{(i)}, \mathbf{u}^{(i)}\}_{i=2}^{N}$

A.2. Gradient Calculation

Below is the missing algorithm to calculate the gradient of bounds $\{\mathbf{l}^{(i)}, \mathbf{u}^{(i)}\}_{i=2}^{N}$ w.r.t. $\boldsymbol{\epsilon}$. We put g in front of a variable to represent its gradient w.r.t. $\boldsymbol{\epsilon}$. Terms like $g[\mathbf{M}]_+$ or $g[\mathbf{M}]_-$ are a bit abused here. It means we put the elements of $g\mathbf{M}$ where the corresponding elements in $\mathbf{M}$ are positive or negative and set the other elements to be 0. In addition, f_1 is the indicator function which returns 1 if input is true and 0 otherwise. When the input is a tensor, the function is applied elementwisely and return a tensor of the same shape.

In **Algorithm** 5, $g\mathbf{D}^{(i)}$, $g\mathbf{m}_1^{(i)}$ and $g\mathbf{m}_2^{(i)}$ can be obtained immediately after the calculation of $\mathbf{D}^{(i)}$, $\mathbf{m}_1^{(i)}$ and $\mathbf{m}_2^{(i)}$. We can run the for-loop in **Algorithm** 5 immediately after the corresponding iteration in **Algorithm** 1. That is to say, we do not need to wait for **Algorithm** 1 to terminate before calculating the gradients like what back-propagation does. This can improve the computational efficiency.

B. Different Bound Estimation Algorithms

It is a bit counterintuitive to find that **Algorithm** 3 and 4 are actually complementary. There is *no guarantee* which one is better in all cases and combining them together in **Algorithm** 1 is a necessary. We use the following two toy examples in Figure 4 to demonstrate the pros and cons of both algorithms.
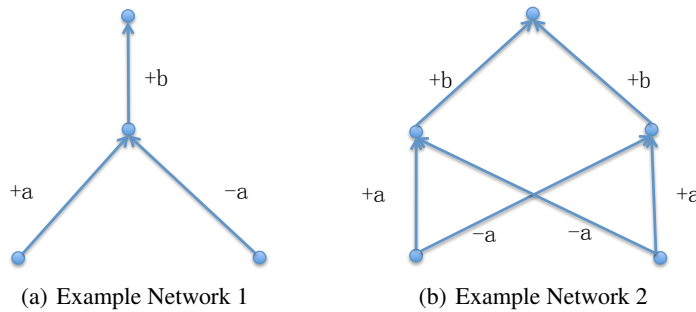

(a) Example Network 1   (b) Example Network 2

Figure 4. Two toy networks.

Algorithm 5 Gradient Calculation

1: Input: Parameters $\{\mathbf{W}^{(i)}, \mathbf{b}^{(i)}\}_{i=1}^{N-1}$, perturbation set $\mathcal{S}_\epsilon(\mathbf{x})$, bounds $\{\mathbf{l}^{(i)}, \mathbf{u}^{(i)}\}_{i=2}^{N}$, values $\{\mathbf{D}^{(i)}, \mathbf{m}_1^{(i)}, \mathbf{m}_2^{(i)}\}_{i=1}^{N}$

2: $g\mathbf{l}^{(2)} = -|\mathbf{W}^{(1)}|$

3: $g\mathbf{u}^{(2)} = |\mathbf{W}^{(1)}|$

4: $g\mathbf{M}^{(1)} = \mathbf{0}$

5: $g\phi^{(2)} = 0$

6: **for** i = 2, ..., N − 1 **do**

7:     Run algorithm 1 to obtain $\mathbf{D}^{(i)}$, $\mathbf{m}_1^{(i)}$ and $\mathbf{m}_2^{(i)}$.

8:     $\mathbf{l}_{naive}^{(i+1)} = \mathbf{W}_+^{(i)}\left[\sigma'(\mathbf{l}^{(i)}) \odot g\mathbf{l}^{(i)}\right] + \mathbf{W}_-^{(i)}\left[\sigma'(\mathbf{u}^{(i)}) \odot g\mathbf{u}^{(i)}\right]$

9:     $\mathbf{u}_{naive}^{(i+1)} = \mathbf{W}_-^{(i)}\left[\sigma'(\mathbf{l}^{(i)}) \odot g\mathbf{l}^{(i)}\right] + \mathbf{W}_+^{(i)}\left[\sigma'(\mathbf{u}^{(i)}) \odot g\mathbf{u}^{(i)}\right]$

10:     $g\mathbf{D}^{(i)} = g\mathbf{l}^{(i)}\frac{\partial \mathbf{D}^{(i)}}{\partial \mathbf{l}^{(i)}} + g\mathbf{u}^{(i)}\frac{\partial \mathbf{D}^{(i)}}{\partial \mathbf{u}^{(i)}}$

11:     $g\mathbf{m}_1^{(i)} = g\mathbf{l}^{(i)}\frac{\partial \mathbf{m}_1^{(i)}}{\partial \mathbf{l}^{(i)}} + g\mathbf{u}^{(i)}\frac{\partial \mathbf{m}_1^{(i)}}{\partial \mathbf{u}^{(i)}}$

12:     $g\mathbf{m}_2^{(i)} = g\mathbf{l}^{(i)}\frac{\partial \mathbf{m}_2^{(i)}}{\partial \mathbf{l}^{(i)}} + g\mathbf{u}^{(i)}\frac{\partial \mathbf{m}_2^{(i)}}{\partial \mathbf{u}^{(i)}}$

13:     $g\mathbf{M}^{(j)} = \mathbf{W}^{(i)}g\mathbf{D}^{(i)}\mathbf{M}^{(j)} + \mathbf{W}^{(i)}\mathbf{D}^{(i)}g\mathbf{M}^{(j)}$ for $j = 1, ..., i-1$

14:     $g\mathbf{M}^{(i)} = \mathbf{0}$

15:     $g\phi^{(i+1)} = \mathbf{W}^{(i)}g\mathbf{D}^{(i)}\phi^{(i)} + \mathbf{W}^{(i)}\mathbf{D}^{(i)}g\phi^{(i)}$

16:     $g\mathbf{l}_{comp}^{(i+1)} = \sum_{j=1}^{i}\left(g[\mathbf{M}^{(j)}]_-\mathbf{m}_2^{(j)} + [\mathbf{M}^{(j)}]_- g\mathbf{m}_2^{(j)} + g[\mathbf{M}^{(j)}]_+\mathbf{m}_1^{(j)} + [\mathbf{M}^{(j)}]_+ g\mathbf{m}_1^{(j)}\right) + g\phi^{(i+1)}$

17:     $g\mathbf{u}_{comp}^{(i+1)} = \sum_{j=1}^{i}\left(g[\mathbf{M}^{(j)}]_-\mathbf{m}_1^{(j)} + [\mathbf{M}^{(j)}]_- g\mathbf{m}_1^{(j)} + g[\mathbf{M}^{(j)}]_+\mathbf{m}_2^{(j)} + [\mathbf{M}^{(j)}]_+ g\mathbf{m}_2^{(j)}\right) + g\phi^{(i+1)}$

18:     $g\mathbf{l}^{(i+1)} = f_{\mathbf{1}}(\mathbf{l}_{naive}^{(i+1)} > \mathbf{l}_{comp}^{(i+1)}) \odot g\mathbf{l}_{naive}^{(i+1)} + f_{\mathbf{1}}(\mathbf{l}_{naive}^{(i+1)} \leq \mathbf{l}_{comp}^{(i+1)}) \odot g\mathbf{l}_{comp}^{(i+1)}$

19:     $g\mathbf{u}^{(i+1)} = f_{\mathbf{1}}(\mathbf{u}_{naive}^{(i+1)} \leq \mathbf{u}_{comp}^{(i+1)}) \odot g\mathbf{u}_{naive}^{(i+1)} + f_{\mathbf{1}}(\mathbf{u}_{naive}^{(i+1)} > \mathbf{u}_{comp}^{(i+1)}) \odot g\mathbf{u}_{comp}^{(i+1)}$

20: **end for**

21: Output: Gradients $\{g\mathbf{l}^{(i)}, g\mathbf{u}^{(i)}\}_{i=2}^{N}$

The activation function in the hidden layer is ReLU, the weights of each connection are shown in Figure 4 and we assume $a, b > 0$. Let the input point be (x, x) and $x > 0$. The adversarial budgets for both features are $\epsilon > 0$. We consider the bounds of both algorithms for both networks as follows.

For network 1, the bounds of the pre-activation for the only hidden neuron are $[-2a\epsilon, +2a\epsilon]$ in both algorithms. *Simple algorithm* obtains $[0, +2a\epsilon]$ as the bound for post-activation and $[0, +2ab\epsilon]$ for the final output. On the other hand, *quadratic algorithm* obtains $\mathbf{D}^{(2)} = [+0.5]$ and $[0]^T \leq \mathbf{m}^{(2)} \leq [+a\epsilon]^T$. Then the expression for the final output is $[+b][+0.5][+a \quad -a]\mathbf{m}^{(1)} + [+b]\mathbf{m}^{(2)} = [+0.5ab \quad -0.5ab]\mathbf{m}^{(1)} + [+b]\mathbf{m}^{(2)}$. As $[-\epsilon \quad -\epsilon]^T \leq \mathbf{m}^{(1)} \leq [+\epsilon \quad +\epsilon]^T$ given by perturbation budget, so the final output bounds of *quadratic algorithm* are $[-ab\epsilon, +2ab\epsilon]$. The true bound for the output in network 1 is $[0, +2ab\epsilon]$, so *simple algorithm* wins in this case.

Similarly, the bounds of the pre-activation for both neurons in hidden layer in network 2 are $[-2a\epsilon, +2a\epsilon]$. *Simple algorithm* obtains $[0, +2a\epsilon]$ as the bound for post-activation and $[0, +4ab\epsilon]$ for the final output. On the other hand, *quadratic algorithm* obtains $\mathbf{D}^{(2)} = diag([+0.5, +0.5])$ and $[0, 0]^T \leq \mathbf{m}^{(2)} \leq [+a\epsilon, +a\epsilon]^T$. The expression for the final output is $[+b \quad +b]\begin{bmatrix} +0.5 & 0 \\ 0 & +0.5 \end{bmatrix}\begin{bmatrix} +a & -a \\ -a & +a \end{bmatrix}\mathbf{m}^{(1)} + [+b \quad +b]\mathbf{m}^{(2)} = [+b \quad +b]\mathbf{m}^{(2)}$ and the final bound is $[0, +2ab\epsilon]$. The true bound for the output in network 2 is also $[0, +2ab\epsilon]$, so *quadratic algorithm* wins in this case.

Obviously, as the combination of both algorithms, **Algorithm** 1 outputs the optimal in both cases.

To summarize, *quadratic algorithm* can, to some extent, capture the composition of transformations in the neural network. For example, the pre-activations of the hidden layer in network 2 are always additive inverse and this constrains the output range of the network. *Quadratic algorithm* can detect this constraint when calculating the output bound, as the first term of the final output's expression cancels out. *Simple algorithm* totally ignores that as it will discard all information of previous layers not directly connected to the current layer. However, *quadratic algorithm* use a linear approximation of the activation function while *simple algorithm* use the exact one. Unnecessary linearization made *quadratic algorithm* obtain suboptimal bound in cases like network 1.

Empirically, *simple algorithm* prefers larger perturbation, because the linearization of activation function invokes larger error here. *Quadratic algorithm* is suitable when the layer size is large, as the composition of large matrix transformation

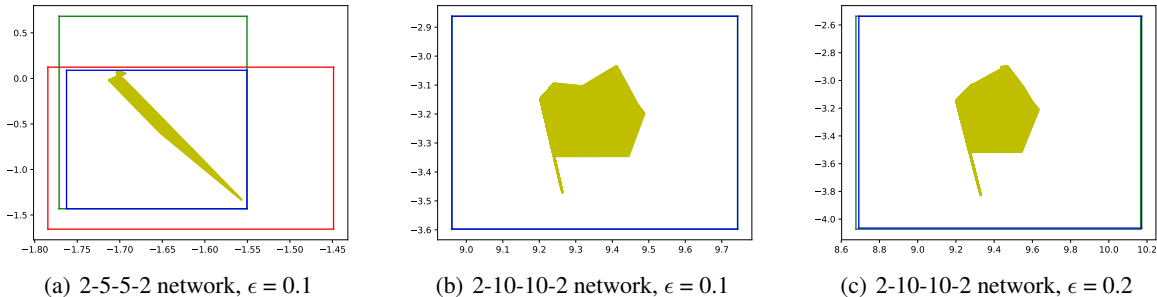typically means more terms can be cancelled out.

(a) 2-5-5-2 network, $\epsilon = 0.1$ (b) 2-10-10-2 network, $\epsilon = 0.1$ (c) 2-10-10-2 network, $\epsilon = 0.2$

Figure 5. Visualization of bounds from *simple* (red), *quadratic* (green) and combined (blue) algorithms with different hidden neurons and uniform bounded perturbation budgets τ. (bounds of *simple algorithm* for (b) and (c) are out of scope.) The set of all possible outputs are in yellow.

Figure 5 demonstrates the different bounds and possible outputs in different toy cases. We generate neural networks with two hidden layers of random weights and compare the bounds obtained by different algorithms. In these cases, we can see the bounds of *simple algorithm* become loose quickly with the increase of layer size while the bounds of *quadratic algorithm* might be the worse when perturbation budgets are large. Of course, as Kolter & Wong (2017) points out, all bounds become looser in larger layer size and larger perturbation budgets.

C. Extra Experiment Results

C.1. Volume of Bounds

Like Figure 2, we put examples of other models and other datasets in Figure 6. We show that the results are consistent: the distributions of non-uniform bounds among input features of the robust models have a 'long tail', which indicates the output logits are affected little by some input features.

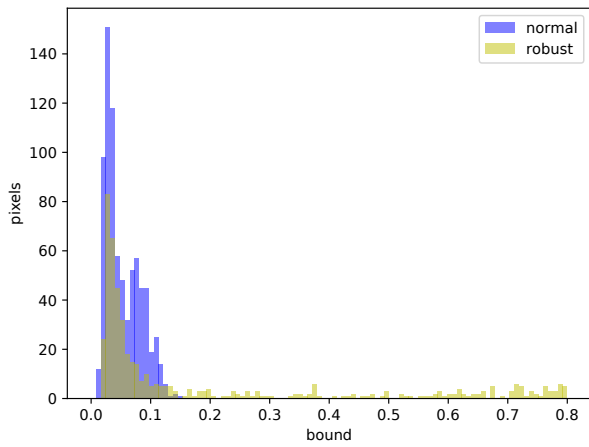### C.2. Robustness and Model Interpretability

Figure 7 gives more examples of bounding maps. Like the model discussed in section 4.2.2, we train two model: one to distinguish digit '3' from digit '8', the other to distinguish digit '1' from digit '2'.
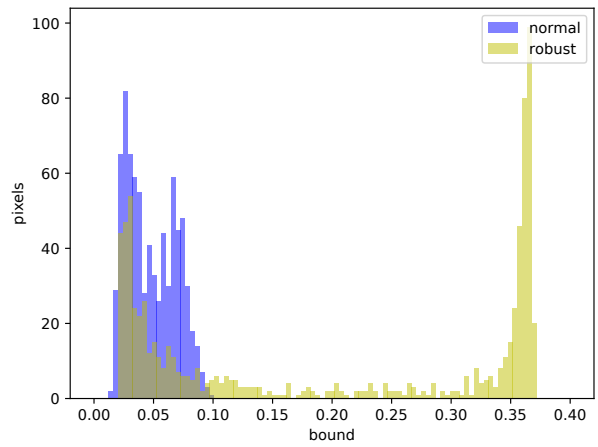
C.3. Robustness and Decision Boundary

We put the average and minimum values of cosine similarity between ϵ values of all image pairs in Table 3.

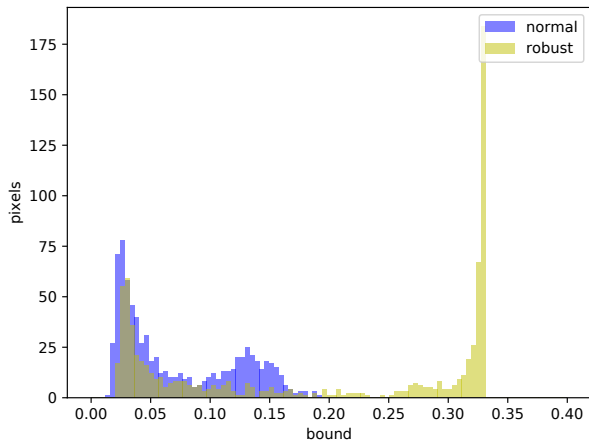| Dataset	Architecture	Adversarial Training	Mean Cosine	Minimum Cosine
MNIST	100-100-100	-	0.9548	0.2304
		PGD, $\tau = 0.1$	0.9957	0.9155
	300-300-300	-	0.9774	0.5038
		PGD, $\tau = 0.1$	0.9964	0.9104
	500-500-500	-	0.9874	0.6367
		PGD, $\tau = 0.1$	0.9941	0.8920
Fashion-MNIST	1024-1024-1024	-	0.9804	0.5257
		PGD, $\tau = 0.1$	0.9931	0.8891
SVHN	1024-1024-1024	-	0.9836	0.7129
		PGD, $\tau = 0.1$	0.9952	0.9339

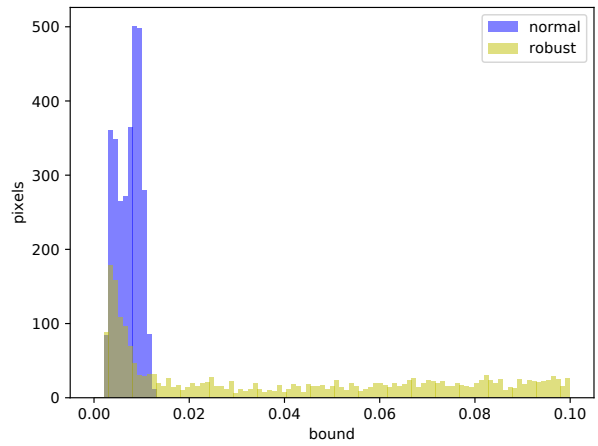*Table 3.* Cosine Similarity of ϵ

(a) '100-100-100' models on MNIST

(b) '500-500-500' models on MNIST

(c) '1024-1024-1024' models on Fashion-MNIST

(d) '1024-1024-1024' models on SVHN

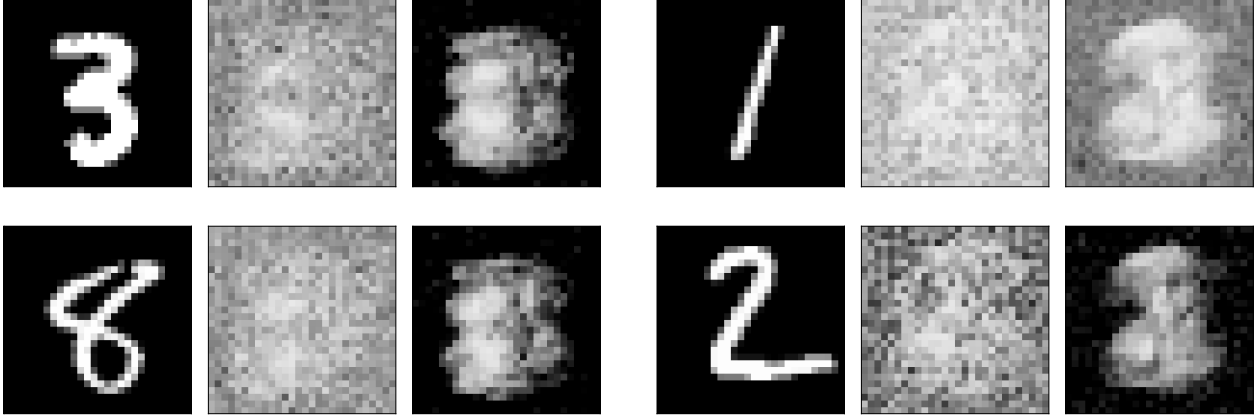*Figure 6.* More examples of distributions of bounds for normal and robust models among all pixels.

Figure 7. Additional examples of bounding maps of images. The images on the left are based on model distinguishing digit '3' from digit '8'. The images on the right are based on model distinguishing digit '1' from digit '2'. Each tuple of 3 images demonstrate the original images, the bounding map of the normal model and the bounding map of the robust model.

D. Algorithm for General Architectures

D.1. General Linear Layers

We first show our formulation for fully-connected layer in **Algorithm** 1 can be naturally generalized to *any linear layer* with fixed variables. Here, we use popular convolutional layer as an example.

Given a convolutional layer with weights $\mathbf{W}^{(c)} \in \mathbb{R}^{n_{in} \times n_{out} \times f_w \times f_h}$, input $x_{in} \in \mathbb{R}^{n_{in} \times w_{in} \times h_{in}}$ and output $x_{out} \in \mathbb{R}^{n_{out} \times w_{out} \times h_{out}}$. If we do not use any padding, then $w_{out} = w_{in} - f_w + 1$ and $h_{out} = h_{in} - f_h + 1$. Therefore, we can reshape x_{in}, x_{out} into one-dimensional vectors x'_{in}, x'_{out} and rewrite convolutional operator in matrix multiplication form $x'_{out} = \mathbf{W}x'_{in}$. $\mathbf{W}$ is a fixed matrix defined as follows where $i_1, i_2, j_1, j_2, k_1, k_2$ are integers.

$$\mathbf{W}_{i_1 w_{in} h_{in} + j_1 h_{in} + k_1, i_2 w_{out} h_{out} + j_2 h_{in} + k_2} = \begin{cases} \mathbf{W}^{(c)}_{i_1, i_2, j_1 - j_2, k_1 - k_2}; 0 \le j_1 - j_2 \le f_w - 1, 0 \le k_1 - k_2 \le f_h - 1 \\ 0 \qquad\qquad\qquad\qquad ; \text{otherwise} \end{cases} \tag{7}$$

As a result, we will use matrix multiplication [4] to represent *any linear layer* defined in the neural networks.

D.2. General Feedforward Neural Networks

As Figure 8 shows, any general feedforward neural networks can be considered as directed acyclic graph (DAG) $G(V, E)$. The vertices $\{1, 2, ..., N - 1, N\}$ represent neurons of each layer and the activation functions are applied on the internal nodes i.e. hidden units. Any edge $(j, i) \in E$ corresponds to a weight matrix $\mathbf{W}^{(j \to i)}$ representing a direct connection between the output of layer j and the input of layer i $(i > j)$.

Now we can define the formulation of a general feedforward neural network, which covers popular residual networks (ResNet) (He et al., 2016) and densely connected networks (DenseNet) (Huang et al., 2017a).

$$\mathbf{z}^{(i)} = \sum_{j=1}^{i-1} \mathbf{W}^{(j \to i)} \hat{\mathbf{z}}^{(j)} + \mathbf{b}^{(i)}$$
$$\hat{\mathbf{z}}^{(i)} = \sigma(\mathbf{z}^{(i)}) \tag{8}$$

Formulation (8) can be reduced to (1) if no skip-connection exists i.e. $\mathbf{W}^{(j \to i)} = 0 \ \forall j \ne i - 1$. For notation simplicity, we first define $\mathbf{P}^{(n)}(j, i)$ as the set of all paths of length n from vertex j to i $(i > j)$ in graph $G(V, E)$:
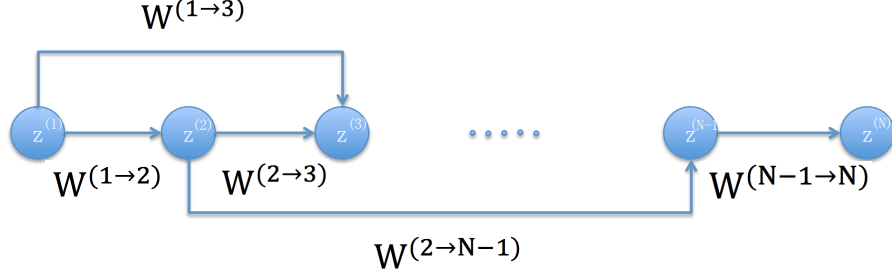
---

[4]We drop bias term for simplicity.

Figure 8. An example of DAG representation of a general feedforward neural networks.

$$\mathbf{P}^{(n)}(j,i) = \{(p_0, p_1, ..., p_{n-1}, p_n) | j = p_0 < p_1 < ... < p_{n-1} < p_n = i, (p_t, p_{t+1}) \in E \; \forall 0 \le t \le n-1\} \tag{9}$$

Then we can define the linearized composite transformation from layer j to layer i as follows:

$$\mathbf{M}^{(j \to i)} = \sum_{n=1}^{i-j} \sum_{\mathbf{p} \in \mathbf{P}^{(n)}(j,i)} \Pi_{t=0}^{n-1} \mathbf{W}^{(p_t \to p_{t+1})} \mathbf{D}^{(p_t)} \quad \forall \, j < i \tag{10}$$

Specially, we define $\mathbf{M}^{(j \to i)} = \mathbf{I} \; \forall j = i$. Similar to equation (4), we can write the output of each layer $\mathbf{z}^{(i)}$ by:

$$\mathbf{z}^{(i)} = \sum_{l=2}^{i} \mathbf{M}^{(l \to i)} \mathbf{W}^{(1 \to l)} \mathbf{x} + \sum_{h=1}^{i} \mathbf{M}^{(h \to i)} \mathbf{b}^{(h)} + \sum_{h=1}^{i-1} \sum_{l=h+1}^{i} \mathbf{M}^{(l \to i)} \mathbf{W}^{(h \to l)} \mathbf{m}^{(h)} \tag{11}$$

In equation (10) and (11), $\{\mathbf{D}^{(h)}, \mathbf{m}^{(h)}\}_{h=1}^{N}$ are defined in the same way as linear approximation in Section 3.1.

Based on equation (11), we can design the bound estimation algorithm (**Algorithm** 6) for a general feedforward neural network. In line D.2, $\mathbf{M}^{(j \to i+1)}$ can be calculated recursively as follows, so the total complexity is still quadratic in matrix multiplication.

$$\mathbf{M}^{(j \to i+1)} = \sum_{h=j+1}^{i} \mathbf{W}^{(h \to i+1)} \mathbf{D}^{(h)} \mathbf{M}^{(j \to h)} \tag{12}$$

We can then follow the same path, gradient calculation and optimization by the augmented Lagragian method, to estimate the certified region of the largest volume for a given data point $\mathbf{x}$.

Algorithm 6 Bound Estimation for General Feedforward Neural Network

---

1: Input: Parameters $\{\mathbf{W}^{(i)}, \mathbf{b}^{(i)}\}_{i=1}^{N-1}$, perturbation set $\mathcal{S}_{\boldsymbol{\epsilon}}(\mathbf{x})$.

2: $\phi^{(1)} = \mathbf{x}, \mathbf{D}^{(1)} = \mathbf{I}, \mathbf{m}_1^{(1)} = -\boldsymbol{\epsilon}, \mathbf{m}_2^{(1)} = \boldsymbol{\epsilon}$

3: $\mathbf{l}^{(2)} = \mathbf{W}^{(1)}\mathbf{x} - \mathbf{W}_+^{(1)}\boldsymbol{\epsilon} + \mathbf{W}_-^{(1)}\boldsymbol{\epsilon} + \mathbf{b}^{(1)}$

4: $\mathbf{u}^{(2)} = \mathbf{W}^{(1)}\mathbf{x} - \mathbf{W}_-^{(1)}\boldsymbol{\epsilon} + \mathbf{W}_+^{(1)}\boldsymbol{\epsilon} + \mathbf{b}^{(1)}$

5: $\mathbf{M}^{(1 \to 2)} = \mathbf{W}^{(1)}$

6: $\phi^{(2)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$

7: **for** i = 2, ..., $N - 1$ **do**

8:      Calculate $\mathbf{D}^{(i)}, \mathbf{m}_1^{(i)}, \mathbf{m}_2^{(i)}$ based on $\mathbf{l}^{(i)}$ and $\mathbf{u}^{(i)}$.

9:      $\mathbf{l}_{simp}^{(i+1)} = \sum_{j=1}^{i} \left(\mathbf{W}_+^{(j \to i+1)} \sigma(\mathbf{l}^{(j)}) + \mathbf{W}_-^{(j \to i+1)} \sigma(\mathbf{u}^{(j)}) \right)$

10:      $\mathbf{u}_{simp}^{(i+1)} = \sum_{j=1}^{i} \left(\mathbf{W}_-^{(j \to i+1)} \sigma(\mathbf{l}^{(j)}) + \mathbf{W}_+^{(j \to i+1)} \sigma(\mathbf{u}^{(j)}) \right)$

11:      Calculate $\mathbf{M}^{(j \to i+1)}$ for $\forall j < i + 1$ according to equation (10).

12:      $\phi^{(i+1)} = \sum_{j=1}^{i+1} \mathbf{W}^{(j \to i+1)} \mathbf{D}^{(j)} \phi^{(j)} + \mathbf{b}^{(i+1)}$

13:      $\mathbf{l}_{quad}^{(i+1)} = \phi^{(i+1)} + \sum_{h=1}^{i-1} \left[\left(\sum_{l=h+1}^{i} \mathbf{M}^{(l \to i)} \mathbf{W}^{(h \to l)} \right)_+ \mathbf{m}_1^{(h)} + \left(\sum_{l=h+1}^{i} \mathbf{M}^{(l \to i)} \mathbf{W}^{(h \to l)} \right)_- \mathbf{m}_2^{(h)} \right]$

14:      $\mathbf{u}_{quad}^{(i+1)} = \phi^{(i+1)} + \sum_{h=1}^{i-1} \left[\left(\sum_{l=h+1}^{i} \mathbf{M}^{(l \to i)} \mathbf{W}^{(h \to l)} \right)_- \mathbf{m}_1^{(h)} + \left(\sum_{l=h+1}^{i} \mathbf{M}^{(l \to i)} \mathbf{W}^{(h \to l)} \right)_+ \mathbf{m}_2^{(h)} \right]$

15:      $\mathbf{l}^{(i+1)} = \max(\mathbf{l}_{simp}^{(i+1)}, \mathbf{l}_{quad}^{(i+1)})$

16:      $\mathbf{u}^{(i+1)} = \min(\mathbf{u}_{simp}^{(i+1)}, \mathbf{u}_{quad}^{(i+1)})$

17: **end for**

18: Output: Bounds $\{\mathbf{l}^{(i)}, \mathbf{u}^{(i)}\}_{i=2}^{N}$