Global Robustness Evaluation of Deep Neural Networks with Provable Guarantees for the Hamming Distance

Wenjie Ruan[1], **Min Wu**[1], **Youcheng Sun**[1], **Xiaowei Huang**[2], **Daniel Kroening**[1] and **Marta Kwiatkowska**[1]
[1]Department of Computer Science, University of Oxford
[2]Department of Computer Science, University of Liverpool
{wenjie.ruan, min.wu, youcheng.sun, kroening, marta.kwiatkowska}@cs.ox.ac.uk,
xiaowei.huang@liverpool.ac.uk

Abstract

Deployment of deep neural networks (DNNs) in safety-critical systems requires provable guarantees for their correct behaviours. We compute the maximal radius of a safe norm ball around a given input, within which there are no adversarial examples for a trained DNN. We define global robustness as an expectation of the maximal safe radius over a test dataset, and develop an algorithm to approximate the global robustness measure by iteratively computing its lower and upper bounds. Our algorithm is the first efficient method for the Hamming (L_0) distance, and we hypothesise that this norm is a good proxy for a certain class of physical attacks. The algorithm is *anytime*, i.e., it returns intermediate bounds and robustness estimates that are gradually, but strictly, improved as the computation proceeds; *tensor-based*, i.e., the computation is conducted over a set of inputs simultaneously to enable efficient GPU computation; and has *provable guarantees*, i.e., both the bounds and the robustness estimates can converge to their optimal values. Finally, we demonstrate the utility of our approach by applying the algorithm to a set of challenging problems.

1 Introduction

Safety certification for DNNs is challenging owing to the lack of symbolic models and formal specifications. An important requirement specification for DNNs is their robustness to input perturbations. DNNs have been shown to suffer from poor robustness because of their susceptibility to *adversarial examples* [Szegedy *et al.*, 2014; Biggio *et al.*, 2013]. These are small modifications to an input, sometimes imperceptible to humans, that make the neural network unstable. As a result, significant effort has been directed towards approaches for crafting adversarial examples [Goodfellow *et al.*, 2015; Papernot *et al.*, 2016; Carlini and Wagner, 2017]. However, these provide *no* formal guarantees, i.e., no conclusion can be drawn on whether adversarial examples remain.

By contrast, recent efforts in the area of automated verification, e.g.,[Huang *et al.*, 2017; Katz *et al.*, 2017; Lomuscio and Maganti, 2017; Narodytska *et al.*, 2017; Dutta *et al.*, 2017; Ruan *et al.*, 2018a; Wu *et al.*, 2018; Gehr *et al.*, 2018; Mirman *et al.*, 2018], have focused on methods that generate adversarial examples, if they exist, and provide rigorous *local* robustness proofs otherwise. This paper proposes a novel approach to quantify the robustness of DNNs that offers a balance between the guaranteed accuracy of the method (thus, a feature so far exclusive to formal approaches) and the computational efficiency of algorithms that search for adversarial examples (without providing any guarantees).

We focus on the Hamming distance (i.e., the L_0 norm), which measures the number of different vector components given two inputs. In the case of images, this is the number of pixels that are different. On the other hand, L_1, L_2 and L_∞ norms compare how much each pixel has changed. The L_0 distance is fast to compute and is an upper bound on the size of an adversarial perturbation [Papernot *et al.*, 2016; Carlini *et al.*, 2017]. However, the non-continuity and non-differentiability of L_0 is a challenge for the discovery of adversarial attacks. Furthermore, the existing safety verification methods that are designed for other distance metrics are not efficient when using the Hamming distance. This includes SMT/SAT-based methods [Katz *et al.*, 2017], MILP-based work [Dutta *et al.*, 2017], exhaustive search or MCTS-based methods [Huang *et al.*, 2017; Wu *et al.*, 2018], and methods based on abstract interpretation [Gehr *et al.*, 2018]. We remark that, while all L_p norms can be used to identify physical adversarial attacks and have a role to play, on their own or in combination, we hypothesise that the Hamming distance is a good proxy for a particular category of realistic physical attacks in which small modifications (e.g., a sticker) are applied to objects that the system is trained to recognise (e.g., a street sign), leaving the rest of the image unchanged.

In this paper we consider the *global* robustness problem, defined as the *expected maximum safe radius* over a (finite) test dataset, which is a generalisation of the local, pointwise robustness problem. The key idea of our approach is to generate sequences of lower and upper bounds for global robustness *simultaneously* for a set of inputs by using tensor-based parallelisation. Thus, global robustness aims to capture the *worst case* local robustness (worst case maximal safety radius) among a set of inputs. The usefulness of global robustness lies in that it quantifies the size of the perturbation that the system can withstand for a set of inputs, instead of just a single input, since it gives both lower and upper bounds on the

maximal safe radius. Intuitively, a lower bound x means that no adversarial example is possible for perturbation of up to x pixel changes, whereas an upper bound x means that x pixel changes are sufficient to attack a network. By considering a distribution over the set of inputs, we can quantify the robustness of a DNN for this input distribution. Our definition is thus a natural extension of the robustness concept of [Peck *et al.*, 2017].

Our method is *anytime*, *tensor-based*, and offers *provable guarantees*. First, the method is *anytime* in the sense that it can return intermediate results, including upper and lower bounds and robustness estimates. We prove that our approach can gradually, but strictly, improve these bounds and estimates as the computation proceeds. Second, it is *tensor-based*. As we are working with a set of inputs, a straightforward approach is to perform robustness evaluation for the inputs individually and to then merge the results. However, this is inefficient, as the set of inputs is large. To exploit the parallelism offered by GPUs, our approach uses tensors. A tensor can formulate a finite set of inputs (e.g., images) into a multi-dimensional array which can be efficiently processed by GPUs in parallel. An algorithm that is well-suited for GPUs uses tensor operations whenever possible. Third, our approach offers *provable guarantees*. We show that the intermediate bounds and robustness estimates converge to optimal values in finite time. Our experimental results suggest that the algorithm converges quickly in practice.

We implement our approach in a tool we name DeepTRE[1] ("Tensor-based Robustness Evaluation for Deep Neural Networks"), and present experiments via five case studies, including global robustness evaluation; competitive L_0 attacks; saliency map generation for model interpretability and local robustness evaluation on ImageNet DNNs including AlexNet, VGG-16/19 and ResNet-50/101; guidance to the design of robust DNN architectures; and test case generation. All applications above require only simple adaptations of our method, e.g., slight modifications of the constraints or objective functions, or adding further constraints. This demonstrates that our tool is flexible enough to deliver a wide range of promising applications.

Owing to space limitations, we focus on describing the core theoretical concepts, tensor-based parallelization and convergence guarantees, and make an extended companion paper available as [Ruan *et al.*, 2018b].

2 Problem Formulation

Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be an N-layer neural network such that, for a given input $x \in \mathbb{R}^n$, $f(x) = (c_1(x), \ldots, c_m(x)) \in \mathbb{R}^m$ represents the confidence values for m classification labels.

Without loss of generality, we normalise the input to $x \in [0, 1]^n$. The output $f(x)$ is usually normalised to be in $[0, 1]^m$ using a softmax layer. We denote the classification label of input x by $cl(f, x) = \arg\max_{j=1,\ldots,m} c_j(x)$. Note that both f and cl can be generalised to apply to a set $\mathtt{T}_0$ of inputs, i.e., $f(\mathtt{T}_0)$ and $cl(f, \mathtt{T}_0)$, in the standard way.

Definition 1 (Maximum Radius of a Safe Norm Ball) *Given a network $f : \mathbb{R}^n \to \mathbb{R}^m$, a distance metric $||\cdot||_D$, an input $x_0 \in \mathbb{R}^n$ and a real number $d \in \mathbb{R}$, a* norm ball *$B(f, x_0, ||\cdot||_D, d)$ is a subspace of $\mathbb{R}^n$ such that $B(f, x_0, ||\cdot||_D, d) = \{x \mid ||x_0 - x||_D \leq d\}$. The number d is called the* radius *of $B(f, x_0, ||\cdot||_D, d)$. A norm ball $B(f, x_0, ||\cdot||_D, d)$ is* safe *if for all $x \in B(f, x_0, ||\cdot||_D, d)$ we have $cl(f, x) = cl(f, x_0)$. Furthermore, if for all $d' > d$ we have that $B(f, x_0, ||\cdot||_D, d')$ is not safe, then d is called the* maximum safe radius*, and denoted by $d_m(f, x_0, ||\cdot||_D)$.*

Intuitively, a norm ball $B(f, x_0, ||\cdot||_D, d)$ includes all inputs whose distance to x_0, measured by $||\cdot||_D$, is within d.

We define the (*global*) robustness evaluation problem over a test dataset $\mathtt{T}$, which is a set of i.i.d. inputs sampled from a distribution μ representing the problem f is working on. We use $|\mathtt{T}|$ to denote the number of inputs in $\mathtt{T}$. When $|\mathtt{T}| = 1$, we call it *local* robustness.

Definition 2 (Robustness Evaluation) *Given a network f, a finite set $\mathtt{T}_0$ of inputs, and $||\cdot||_D$,* robustness evaluation*, denoted as $R(f, \mathtt{T}_0, ||\cdot||_D)$, is an optimisation problem:*

$$\min_{\mathtt{T}} ||\mathtt{T}_0 - \mathtt{T}||_D \quad \textit{s.t.} \quad cl(f, x_i) \neq cl(f, x_{0,i}) \tag{1}$$

for all $i \in \{1, \ldots, |\mathtt{T}_0|\}$ where $\mathtt{T} = (x_i)_{i=1,\ldots,|\mathtt{T}_0|}$ and $\mathtt{T}_0 = (x_{0,i})_{i=1,\ldots,|\mathtt{T}_0|}$.

Intuitively, we aim to find a minimum distance between the original set $\mathtt{T}_0$ and a new, homogeneous set[2] $\mathtt{T}$ of inputs such that all inputs in $\mathtt{T}_0$ are mis-classified.

L_0 Norm The distance metric $||\cdot||_D$ can be any mapping $||\cdot||_D : \mathbb{R}^n \times \mathbb{R}^n \to [0, \infty]$ that satisfies the metric conditions. In this paper, we focus on the L_0 norm[3]. For two inputs x_0 and x, their L_0 distance, denoted as $||x - x_0||_0$, is the number of their components that are different. When working with a test dataset $\mathtt{T}_0$ (all inputs in $\mathtt{T}_0$ are i.i.d.), we define

$$\begin{aligned} ||\mathtt{T} - \mathtt{T}_0||_0 &= \mathtt{E}_{x_0 \in \mathtt{T}_0}[||x - x_0||_0] \qquad \text{(our definition)} \\ &= \frac{1}{|\mathtt{T}_0|} \sum_{x_0 \in \mathtt{T}_0} ||x - x_0||_0 \end{aligned} \tag{2}$$

where $x \in \mathtt{T}$ is a homogeneous input to $x_0 \in \mathtt{T}_0$. While other norms such as L_1, L_2 and L_∞ have been widely applied for generating adversarial examples [Papernot *et al.*, 2016; Kurakin *et al.*, 2016], studies of robustness evaluation for DNNs based on the L_0 norm are few and far between [Papernot *et al.*, 2016; Carlini *et al.*, 2017; Wicker *et al.*, 2018; Huang *et al.*, 2017; Wu *et al.*, 2018]. In Appendix B of [Ruan *et al.*, 2018b], we discuss why L_0 is an important distance metric from various perspectives.

3 Anytime Robustness Evaluation

The evaluation of robustness following Definition 2 is hard for the L_0-norm. In Appendix A.1 of [Ruan *et al.*, 2018b], we discuss its computational complexity and prove that the problem is NP-hard. We propose to compute lower and upper bounds on robustness, and then gradually, but *strictly*, improve the

[1] Available on GitHub: https://github.com/TrustAI/L0-TRE.

[2] Two sets $\mathtt{T}_0$ and $\mathtt{T}$ are homogeneous if they have the same number of elements and are of the same type.

[3] Strictly speaking, L_0 is not a norm, but it is commonly referred to as such.

bounds so that the gap between them can eventually be closed. Although in practice run times can be long, this *anytime* approach provides pragmatic means to make progress. Section 5 shows that our approach achieves *tight* bounds *efficiently* on difficult instances.

Definition 3 (Sequences of Bounds) *Given a robustness evaluation problem $R(f, \mathtt{T}_0, ||\cdot||_D)$, a sequence $\mathcal{L}(\mathtt{T}_0) = \{l_1, l_2, \ldots, l_k\} \in \mathbb{R}$ is an* incremental lower bound sequence *if, for all $1 \leq i < j \leq k$, we have $l_i \leq l_j \leq R(f, \mathtt{T}_0, ||\cdot||_D)$. The sequence is strict, denoted as $\mathcal{L}_s(\mathtt{T}_0)$, if for all $1 \leq i < j \leq k$, we have either $l_i < l_j$ or $l_i = l_j = R(f, \mathtt{T}_0, ||\cdot||_D)$. Similarly, we can define a* decremental upper bound sequence $\mathcal{U}(\mathtt{T}_0)$ *and a strict decremental upper bound sequence $\mathcal{U}_s(\mathtt{T}_0)$.*

We will, in Section 4, introduce our algorithms for computing these two sequences of lower and upper bounds. For now, assume they exist, then at a certain time $t > 0$,

$$l_t \leq R(f, \mathtt{T}_0, ||\cdot||_D) \leq u_t \tag{3}$$

holds.

Definition 4 (Anytime Robustness Evaluation) *Based on two given bounds $[l_t, u_t]$, we define its centre and radius as follows.*

$$U_c(l_t, u_t) = \frac{1}{2}(l_t + u_t) \quad \textit{and} \quad U_r(l_t, u_t) = \frac{1}{2}(u_t - l_t) \tag{4}$$

The anytime evaluation of $R(f, \mathtt{T}_0, ||\cdot||_D)$ at time *t, denoted as $R_t(f, \mathtt{T}_0, ||\cdot||_D)$, is the pair $(U_c(l_t, u_t), U_r(l_t, u_t))$.*

The anytime evaluation will be returned whenever the computational procedure is interrupted. Intuitively, we use the centre $U_c(l_t, u_t)$ to represent the current estimate, and the radius $U_r(l_t, u_t)$ to represent its error bound. Essentially, we can bound the true robustness $R(f, \mathtt{T}_0, ||\cdot||_D)$ via the anytime robustness evaluation.

4 Tensor-based Algorithms for Upper and Lower Bounds

We present our approach to generate the sequence of bounds. For both the lower bounds and the upper bounds, we need the following definition.

Definition 5 (Complete Set of Subspaces for an Input) *Given an input $x_0 \in [0,1]^n$ and a set of t dimensions $T \subseteq \{1, ..., n\}$ such that $|T| = t$, the* subspace *for x_0, denoted by $X_{x_0,T}$, is a set of inputs $x \in [0,1]^n$ such that $x(i) \in [0,1]$ for $i \in T$ and $x(i) = x_0(i)$ for $i \in \{1, ..., n\} \setminus T$. Furthermore, given an input $x_0 \in [0,1]^n$ and a number $t \leq n$, we define*

$$\mathcal{X}(x_0, t) = \{X_{x_0,T} \mid T \subseteq \{1, ..., n\}, |T| = t\} \tag{5}$$

as the complete *set of subspaces for input x_0.*

Intuitively, elements in $X_{x_0,T}$ share the same value with x_0 on the dimensions other than T, and may take any legal value for the dimensions in T. Moreover, $\mathcal{X}(x_0, t)$ includes all sets $X_{x_0,T}$ for any possible combination T with t dimensions.

Next, we define the subspace sensitivity for a subspace w.r.t. a neural network f, an input x_0 and a test dataset $\mathtt{T}_0$. Recall that $f(x) = (c_1(x), \ldots, c_m(x))$.

Definition 6 (Subspace Sensitivity) *Given an input subspace $X \subseteq [0,1]^n$, an input $x_0 \in [0,1]^n$ and a label j, the* subspace sensitivity *w.r.t. X, x_0, and j is defined as*

$$S(X, x_0, j) = c_j(x_0) - \inf_{x \in X} c_j(x). \tag{6}$$

Let t be an integer. We define the subspace sensitivity for $\mathtt{T}_0$ and t as

$$S(\mathtt{T}_0, t) = (S(X_{x_0}, x_0, j_{x_0}))_{X_{x_0} \in \mathcal{X}(x_0,t), x_0 \in \mathtt{T}_0} \tag{7}$$

where $j_{x_0} = \arg\max_{i \in \{1,\ldots,m\}} c_i(x_0)$ is the classification label of x_0 by network f.

Intuitively, $S(X, x_0, j)$ is the maximal decrease of confidence value of the output label j that can be witnessed from the set X, and $S(\mathtt{T}_0, t)$ is the two-dimensional array of the maximal decreases of confidence values of the classification labels for all subspaces in $\mathcal{X}(x_0, t)$ and all inputs in $\mathtt{T}_0$. It is not hard to see that $S(X, x_0, j) \geq 0$.

Given a test dataset $\mathtt{T}_0$ and an integer $t > 0$, the number of elements in $S(\mathtt{T}_0, t)$ is in $O(|\mathtt{T}_0| \cdot n^t)$, i.e., polynomial in $|\mathtt{T}_0|$ and exponential in t. Note that, by Equation (6), every element in $S(\mathtt{T}_0, t)$ represents an optimisation problem. That is, for $\mathtt{T}_0$, a set of 20 MNIST images, and $t = 1$, this would be $28 \times 28 \times 20 = 15{,}680$ one-dimensional optimisation problems. In the next section, we give a tensor-based formulation and an algorithm to solve this challenging problem via GPU parallelisation.

4.1 Tensor-based Parallelisation for Computing Subspace Sensitivity

A tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ in an N-dimensional space is a mathematical object that has $\prod_{m=1}^{N} I_m$ components and obeys certain transformation rules. Intuitively, tensors are generalisations of vectors (i.e., one index) and matrices (i.e., two indices) to an arbitrary number of indices. Many state-of-the-art deep learning libraries, such as Tensorflow and pyTorch, are utilising tensors to parallelise the computation with GPUs. However, it is nontrivial to write an algorithm working with tensors owing to the limited set of operations on tensors.

The basic idea of our algorithm is to transform a set of non-linear, non-convex optimisation problems as given in Equation (7) into a tensor formulation, and solve a set of optimisation problems via a few DNN queries. First, we introduce the following operations on tensors that are used in our algorithm.

Definition 7 (Mode-n Unfolding and Folding) *Given a tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, the* mode-$n$ unfolding *of $\mathcal{T}$ is a matrix $\mathbf{U}_{[n]}(\mathcal{T}) \in \mathbb{R}^{I_n \times I_M}$ such that $M = \prod_{k=1, k \neq n}^{N} I_k$ and $\mathbf{U}_{[n]}(\mathcal{T})$ is defined by the mapping from element $(i_1, \ldots, i_N)$ to (i_n, j), with*

$$j = \sum_{k=1, k \neq n}^{N} \left[i_k \times \prod_{m=k+1, m \neq n}^{N} I_m \right]. \tag{8}$$

Similarly, the tensor folding $\mathbf{F}$ *folds an unfolded tensor back from a matrix into a full tensor. Tensor unfolding and folding are dual operations and link tensors and matrices.*

Given a neural network f, a number t and a test dataset $\mathtt{T}_0$, each $x_i \in \mathtt{T}_0$ generates a complete set $\mathcal{X}(x_i, t)$ of subspaces. Let $|\mathtt{T}_0| = p$ and $|\mathcal{X}(x_i, t)| = k$. Note that, for different x_i and x_j, we have $|\mathcal{X}(x_i, t)| = |\mathcal{X}(x_j, t)|$. Given an error tolerance $\epsilon > 0$, by applying grid search, we can recursively sample $\Delta = 1/\epsilon$ numbers in each dimension, and turn each subspace $X_{x_i} \in \mathcal{X}(x_i, t)$ into a two-dimensional grid $\mathcal{G}(X_{x_i}) \in \mathbb{R}^{n \times \Delta^t}$. We thus can formulate the following tensor:

$$\mathcal{T}(\mathtt{T}_0, t) = \text{Tensor}((\mathcal{G}(X_{x_i}))_{x_i \in \mathtt{T}_0, X_{x_i} \in \mathcal{X}(x_i,t)}) \in \mathbb{R}^{n \times \Delta^t \times p \times k} \tag{9}$$

In Appendix A.2 of [Ruan *et al.*, 2018b], we show that grid search provides the guarantee of reaching the global minimum by utilising the Lipschitz continuity in DNNs.

Then, we apply the mode-1 tensor unfolding operation to have $\mathbf{T}_{[1]}(\mathcal{T}(\mathtt{T}_0, t)) \in \mathbb{R}^{n \times M}$ such that $M = \Delta^t \cdot p \cdot k$. Then this tensor can be fed into the DNN f to obtain

$$Y(\mathtt{T}_0, t) = f(\mathbf{T}_{[1]}(\mathcal{T}(\mathtt{T}_0, t))) \in \mathbb{R}^M. \tag{10}$$

After computing $Y(\mathtt{T}_0, t)$, we apply a tensor folding operation to obtain

$$\mathcal{Y}(\mathtt{T}_0, t) = \mathbf{F}(Y(\mathtt{T}_0, t)) \in \mathbb{R}^{\Delta^t \times p \times k}. \tag{11}$$

Here, we should note the difference between $\mathbb{R}^{\Delta^t \cdot p \cdot k}$ and $\mathbb{R}^{\Delta^t \times p \times k}$, with the former being a one-dimensional array and the latter a tensor. On $\mathcal{Y}(\mathtt{T}_0, t)$, we search the minimum values along the first dimension to obtain[4]

$$V(\mathtt{T}_0, t)_{\min} = \min(\mathcal{Y}(\mathtt{T}_0, t), 1) \in \mathbb{R}^{p \times k}. \tag{12}$$

Thus, we have now solved all $p \times k$ optimisation problems. We then construct the tensor

$$V(\mathtt{T}_0, t) = (\overbrace{c_{j_{x_i}}(x_i), ..., c_{j_{x_i}}(x_i)}^{k})_{x_i \in \mathtt{T}_0} \in \mathbb{R}^{p \times k} \tag{13}$$

from the set $\mathtt{T}_0$. Recall that $j_{x_i} = \arg\max_{k \in \{1,\ldots,m\}} c_k(x_i)$. Intuitively, $V(\mathtt{T}_0, t)$ is the tensor that contains the starting points of the optimisation problems and $V(\mathtt{T}_0, t)_{\min}$ the resulting optimal values. The following theorem shows the correctness of our computation, where $S(\mathtt{T}_0, t)$ has been defined in Definition 6.

Theorem 1 *Let $\mathtt{T}_0$ be a test dataset and t an integer. We have $S(\mathtt{T}_0, t) = V(\mathtt{T}_0, t) - V(\mathtt{T}_0, t)_{\min}$.*

We remark that we only need a single DNN query in Equation (10) to perform the computation above.

4.2 Tensor-based Parallelisation for Computing Lower and Upper Bounds

Let $\mathcal{S}(\mathtt{T}_0, t) \in \mathbb{R}^{n \times p \times k}$ be the tensor obtained by replacing every element in $S(\mathtt{T}_0, t)$ by its corresponding input that, according to the computation of $V(\mathtt{T}_0, t)_{\min}$, causes the largest decreases of the confidence values of the classification labels. We call $\mathcal{S}(\mathtt{T}_0, t)$ the *solution tensor* of $S(\mathtt{T}_0, t)$. The computation of $\mathcal{S}(\mathtt{T}_0, t)$ can be done using very few tensor operations over $\mathcal{T}(\mathtt{T}_0, t)$ and $\mathcal{Y}(\mathtt{T}_0, t)$, which have been given in Section 4.1. We omit the details.

[4]Here we use the Matlab notation $\min(Y, k)$, which computes the minimum values over the k-th dimension for a multi-dimensional array Y. We use similar notation in the remainder of the paper.

Lower Bounds

We reorder $S(\mathtt{T}_0, t)$ and $\mathcal{S}(\mathtt{T}_0, t)$ w.r.t. decreasing values in $S(\mathtt{T}_0, t)$. Then, we retrieve the first row of the third dimension in tensor $\mathcal{S}(\mathtt{T}_0, t)$, i.e., $\mathcal{S}(\mathtt{T}_0, t)[:, :, 1] \in \mathbb{R}^{n \times p}$, and check whether $cl(f, \mathcal{S}(\mathtt{T}_0, t)[:, :, 1]) = cl(f, \mathtt{T}_0)$. The result is an array of Boolean values, each of which is associated with an input $x_i \in \mathtt{T}_0$. If any element associated with x_i in the resulting array is *false*, we conclude that $d_m(f, x_i, ||\cdot||_D) = t - 1$, i.e., the maximum safe radius has been obtained and the computation for x_i has converged. On the other hand, if the element associated with x_i is *true*, we update the lower bound for x_i to t. After computing $\mathcal{S}(\mathtt{T}_0, t)$, no further DNN query is needed to compute the lower bounds.

Upper Bounds

The upper bounds are computed by iteratively applying perturbations based on the matrix $\mathcal{S}(\mathtt{T}_0, t)$ for every input in $\mathtt{T}_0$ until a misclassification occurs. However, doing this sequentially for all inputs would be inefficient, since we need to query the network f after every perturbation on each image.

We present an efficient tensor-based algorithm, which enables GPU parallelisation. The key idea is to construct a new tensor $\mathcal{N} \in \mathbb{R}^{n \times p \times k}$ to maintain all the accumulated perturbations over the original inputs $\mathtt{T}$.

- Initialisation: $\mathcal{N}[:, :, 1] = \mathcal{S}(\mathtt{T}_0, t)[:, :, 1]$.
- Iteratively construct the i-th row until $i = k$:

$$\mathcal{N}[:, :, i] = \{\mathcal{N}[:, :, i-1] \boxminus \{\mathcal{N}[:, :, i-1] \Cap \mathcal{S}(\mathtt{T}_0, t)[:, :, i]\}\}$$
$$\Cup \{\mathcal{S}(\mathtt{T}_0, t)[:, :, i] \boxminus \{\mathcal{N}[:, :, i-1] \Cap \mathcal{S}(\mathtt{T}_0, t)[:, :, i]\}\}$$

where $\boxminus$, $\Cap$, and $\Cup$ are tensor operations: $\mathcal{N}_1 \boxminus \mathcal{N}_2$ removes the corresponding non-zero elements in $\mathcal{N}_2$ from $\mathcal{N}_1$; further, $\mathcal{N}_1 \Cap \mathcal{N}_2$ retains those elements that have the same values and sets the other elements to 0; finally, $\mathcal{N}_1 \Cup \mathcal{N}_2$ merges the non-zero elements from two tensors. The two operands of these operations are required to have the same type. Intuitively, $\mathcal{N}[:, :, i]$ represents the result of applying the first i perturbations recorded in $\mathcal{S}(\mathtt{T}_0, t)[:, :, 1 : i]$, which contains the perturbations up to index $i - 1$ (i.e., $\mathcal{N}_{:,i,:}$) plus the new perturbation recorded in $\tilde{\mathcal{X}}_{:,i,:}$.

Subsequently, we unfold $\mathcal{N}$ and pass the result to the DNN f, which yields the classification labels $Y(\mathbf{U}_{[1]}(\mathcal{N})) \in \{1, \ldots, m\}^{p \cdot k}$. After that, a tensor folding operation is applied to obtain $\mathcal{Y}(\mathbf{U}_{[1]}(\mathcal{N})) \in \{1, \ldots, m\}^{p \times k}$. Then we do the tensor folding operation to obtain $C = \text{fold}(Y_c) \in \mathbb{R}^{K \times P}$. Finally, we can compute the minimum column index along each row such that a misclassification occurs, denoted by $\{m_1, ..., m_p\}$ such that $1 \leq m_i \leq k$. Then we let

$$\mathtt{T} = \{\mathcal{N}_{:,i,m_i} \in \mathbb{R}^{n \times p} \mid x_i \in \mathtt{T}_0\}, \tag{14}$$

which is the optimal set of inputs as required in Definition 2.

After computing $\mathcal{S}(\mathtt{T}_0, t)$, we only need one further DNN query to obtain all upper bounds for a given test dataset $\mathtt{T}_0$.

Tightening the Upper Bounds

There may be redundancies in $\mathtt{T} - \mathtt{T}_0$, i.e., not all the changes in $\mathtt{T} - \mathtt{T}_0$ are necessary to observe a misclassification. We therefore remove the redundancies and thereby tighten the

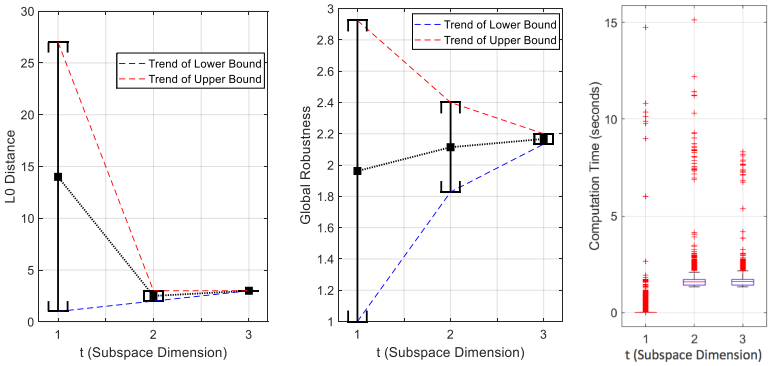


Figure 1: (Left) Convergence of lower bound, upper bound, and estimation of d_m for one image; (Middle) That of global robustness; (Right) Boxplots of the computational time.

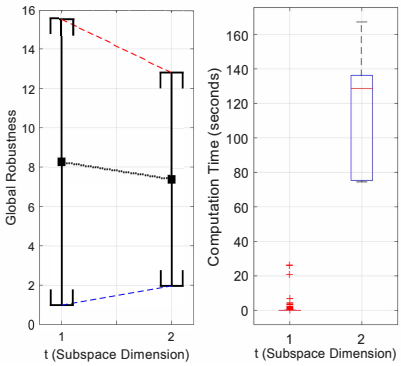


Figure 2: Robustness evaluation of DNN-0 for $t \in \{1, 2\}$ and box-plots of computation time.

upper bounds. We reduce the tightening problem to an optimisation problem similar to that of Definition 2, which enables us to reuse the tensor-based algorithms given above. Assume that $x_{0,i}$ and x_i are two corresponding inputs in $\mathtt{T}$ and $\mathtt{T}_0$, respectively, for $i \in \{1, \ldots, |\mathtt{T}_0|\}$. By abuse of notation, we let $z_{0,i} = x_{0,i} - x_i$ be the part of $x_{0,i}$ on which $x_{0,i}$ and x_i are different, and $l_{0,i} = x_i \Cap x_{0,i}$ be the part of $x_{0,i}$ on which $x_{0,1}$ and x_i are the same. Therefore, $x_{0,i} = z_{0,i} \Cup l_{0,i}$.

Definition 8 (Tightening the Upper Bounds) *Given a network f, a finite test dataset $\mathtt{T}_0$ with their upper bounds $\mathtt{T}$, and $||\cdot||_D$, the tightening problem is an optimisation problem:*

$$\min_{\mathtt{L}_1} ||\mathtt{L}_0 - \mathtt{L}_1||_D \;\; s.t. \;\; cl(f, z_{0,i} \Cup l_{1,i}) \neq cl(f, z_{0,i} \Cup l_{0,i})$$

where $i \in \{1, \ldots, |\mathtt{L}_0|\}$, $\mathtt{L}_0 = (l_{0,i})_{i=1\ldots|\mathtt{T}_0|}$, $\mathtt{L}_1 = (l_{1,i})_{i=1\ldots|\mathtt{L}_0|}$, and $l_{1,i}, l_{0,i} \in [0,1]^{|\mathtt{L}_0|}$. To solve this optimisation problem, we can re-use the tensor-based algorithm for computing lower bounds with minor modifications to the DNN query: before querying the DNN, we apply the $\Cup$ operation to merge with $z_{0,i}$ as in the above equation. Owing to space limitations, we provide the convergence analysis in Appendix A.2 of [Ruan *et al.*, 2018b].

5 Experimental Results

We report experimental evidence for the utilit of our algorithm. Some experiments require simple modifications of the optimisation problem given in Definition 2, e.g., small changes to the constraints. No significant modification to our algorithm is needed to process these variants. In this section, we use five case studies to demonstrate the broad applicability of our tool.

5.1 Convergence Analysis and Global Robustness Evaluation

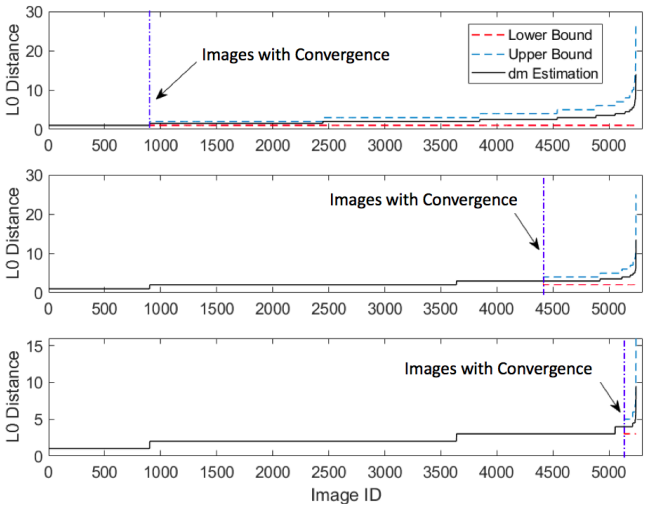
We study the convergence and running time of our anytime global robustness evaluation algorithm on several DNNs in terms of the L_0-norm. To the best of our knowledge, no baseline method exists for this case study. Adversarial attack algorithms for the L_0 norm, which we compare against in Section 5.2, cannot perform robustness evaluation based on both lower and upper bounds with provable guarantees.

We train two DNNs on the MNIST dataset. DNN-0 is trained on the original images with size 28×28 and sDNN on images resized to 14×14. The latter is less robust and used here for the purpose of discussing the convergence of our method. The DNN models are given in Appendix D of [Ruan *et al.*, 2018b], together with training and accuracy statistics. For DNN-0, we work with a set of 2,400 randomly sampled images, and for sDNN we use a set of 5,300 images. We perform the computation on a PC with I7-7700HQ CPU, 16 GB of RAM and NVIDIA GTX-1050Ti GPU.

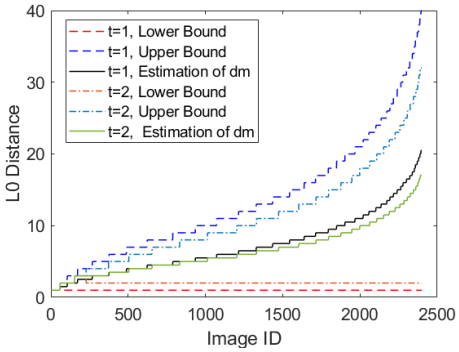
sDNN: Convergence and Robustness Evaluation

Figure 1 (Left) illustrates the speed of convergence of lower and upper bounds and the estimate for d_m (i.e., the maximum safe radius) for an image with a large initial upper bound at L_0 distance 27. This image is chosen to demonstrate the *worst case* for our approach. Working with a single image (i.e., local robustness) is the special case of our optimisation problem where $|\mathtt{T}| = 1$. We choose one of the worst examples among our dataset since the initial upper bound computed is very large: The L_0 distance to the original image is 27. We observe that, when transitioning from $t = 1$ to $t = 2$, the uncertainty radius $U_r(l_t, u_t)$ of d_m is reduced significantly from 26 to 1. Figure 1 (Middle) illustrates the speed of convergence of the global robustness evaluation on the test dataset: our method obtains tight lower and upper bounds efficiently and converges quickly. Notably, we have $U_c(l_t, u_t) = 1.97$ at $t = 1$; the final global robustness is 2.1, so the relative error of the global robustness at $t = 1$ is $< 7\%$. The estimate at $t = 1$ can be obtained in polynomial time, and thus the results demonstrate that our approach is able to provide a good approximation with reasonable error at very low computational cost. Figure 1 (Right) gives the box-plots of the computational time required for individual iterations (i.e., subspace dimension t). We remark that at $t = 1$ it takes less than 0.1 s to process one image, which suggests that the algorithm has potential for real-time applications.

In Figure 3 (a), we plot the upper and lower bounds as well as the estimate for d_m for all tested images. The images are ordered using their upper bounds at $t = 1$. The dashed blue line indicates that all images to the left of this line have converged. The charts show a clear overall trend: our algorithm converges for most images after a few iterations.



(a) sDNN



(b) DNN-0

Figure 3: (a) sDNN: Upper bounds, lower bounds, and estimations of d_m for all sampled images for $t \in \{1, 2, 3\}$ ordered from the top to bottom. (b) DNN-0: Upper bounds, lower bounds, and estimations of d_m for all sampled input images for $t \in \{1, 2\}$.

DNN-0: Global Robustness Evaluation

Figure 3 (b) illustrates the convergence trends for all 2,400 images for a large DNN. We observe that, even for a DNN with tens of thousands of hidden neurons, DeepTRE still achieves tight estimates for d_m for most images. Figure 2 gives the results of anytime global robustness evaluation at $t = 1$ and $t = 2$ for DNN-0, which demonstrates the efficiency of our approach for anytime global robustness evaluation on DNNs. Figure 14 in Appendix D of [Ruan *et al.*, 2018b] features some ground-truth adversarial images[5] returned by our upper bound algorithm.

5.2 Competitive L_0 Attacks

While the generation of attacks is not the primary goal of our method, we observe that our upper bound generation method is highly competitive with state-of-the-art methods for the computation of adversarial images in terms of the L_0 distance. We train MNIST and CIFAR-10 DNNs and compare with JSMA [Papernot *et al.*, 2016], C&W [Carlini and Wagner, 2017], DLV [Huang *et al.*, 2017], SafeCV [Wicker *et al.*, 2018] and DeepGame [Wu *et al.*, 2018], on 1,000 test images. Details of the experimental setup are given in Appendix E of [Ruan *et al.*, 2018b].

Adversarial L_0 Distance

Figure 4 depicts the average and standard deviations of L_0 distances of the adversarial images produced by the five methods. A smaller L_0 distance indicates an adversarial example closer to the original image. For MNIST, the performance of our method is better than JSMA, DLV, and SafeCV, and comparable to C&W and DeepGame. For CIFAR-10, the bar chart reveals that our tool DeepTRE achieves the smallest L_0 distance (modifying 2.62 pixels on average) among all competitors. For this experiment, we stop at $t = 1$ without performing further iterations.

Computational Cost

Figure 5 (note in log-scale) gives running times. Our tensor-based parallelisation method delivers extremely efficient attacks. For example, for MNIST, our method is $18\times$, $100\times$, $1050\times$, and $357\times$ faster than JSMA, C&W, DLV, and SafeCV, respectively. Figure 6 shows that the tensor-based parallelisation significantly improves the computational efficiency: 38 times faster on MNIST DNN and 93 times faster on CIFAR-10 DNN[6]. Appendix E of [Ruan *et al.*, 2018b] compares some of the adversarial examples found by the five methods. The examples illustrate that the modification of one to three pixels suffices to trigger a misclassification even using a well-trained neural network.

5.3 Local Robustness Evaluation for ImageNet

We apply our method to five state-of-the-art ImageNet DNN models, including AlexNet (8 layers), VGG-16 (16 layers), VGG-19 (19 layers), ResNet50 (50 layers), and ResNet101 (101 layers). We set $t = 1$ and generate the lower/upper bounds and estimates of local robustness for an input image. Figure 8 gives the local robustness estimates and their bounds for these networks. The adversarial images on the upper boundaries are featured in the top row of Figure 7. For AlexNet, on this specific image, DeepTRE is able to find its ground-truth adversarial example (local robustness converges at L_0 = 2). We also observe that, for this image, the most robust model is VGG-16 (local robustness = 15) and the most vulnerable one is AlexNet (local robustness = 2). Figure 8 also reveals that, for similar network structures such as VGG-16 and VGG-19, ResNet50 and ResNet101, a model with deeper layers is less robust to adversarial perturbations.

As a byproduct, our method can also generate a saliency map for each input image as shown by the bottom row of Figure 7. Details of the experimental setup are given in Appendix F of [Ruan *et al.*, 2018b].

[5]Ground-truth adversarial images are images at the boundary of a safe norm ball, which was proposed in [Carlini *et al.*, 2017].

[6]The hardware setups are as follows. CPU-1: Tensorflow on an i5-4690S CPU; GPU-1: Tensorflow with parallelisation on an NVIDIA GTX TITAN GPU. CPU-2: Deep Learning Toolbox (Matlab2018b) on an i7-7700HQ CPU; GPU-2: Deep Learning Toolbox (Matlab2018b) with parallelisation on an NVIDIA GTX-1050Ti GPU.

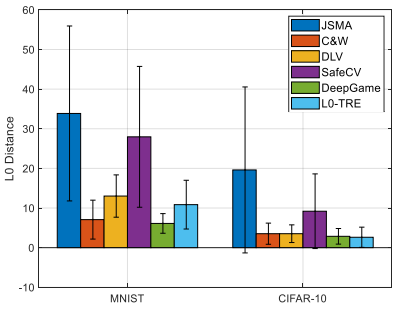


Figure 4: Means and standard deviations of the adversarial L_0 distance.

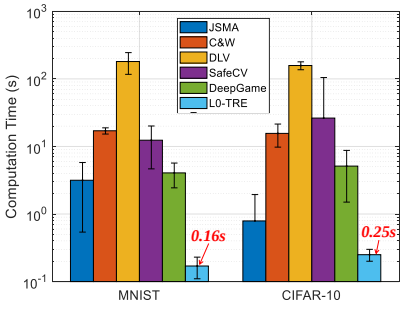


Figure 5: Means and standard deviations of computational time of all methods.

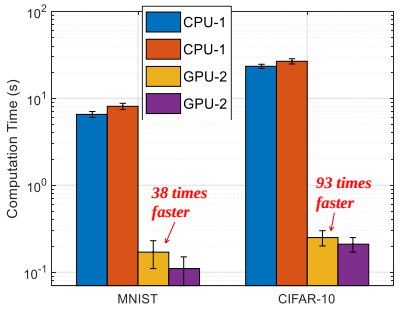


Figure 6: Means and standard deviations of computational time with CPUs or GPUs.

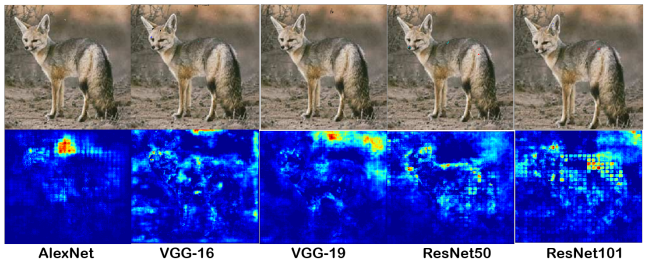


Figure 7: Adversarial examples on upper boundaries (top) and saliency maps (bottom).

5.4 Guiding the Design of Robust DNN Architectures

In this case study, we show how to use our tool DeepTRE for guiding the design of robust DNN architectures.

We trained seven DNNs, named DNN-i for $i \in \{1, \ldots, 7\}$, on the MNIST dataset using the same hardware and software platform and identical training parameters. The DNNs differ in their architecture, i.e., the number of layers and the types of the layers. The architecture matters: while all DNNs achieve 100% accuracy during training, we observe accuracy down to 97.75% on the test dataset. Details of the models are in Appendix G of [Ruan *et al.*, 2018b]. We aim to identify architectural choices that affect robustness.

Figure 9 gives the estimates for global robustness, and their upper and lower bounds at $t = 1$ and $t = 2$ for all seven DNNs. Figure 10 illustrates the means and standard deviations of the d_m estimates and the uncertainty radius for all 1,000 sampled testing images. We also find that the local robustness (i.e., robustness evaluated on a single image) of a network is coincident with its global robustness, and so is the uncertainty radius.

We observe the following: *i)* number of layers: a very deep DNN (i.e., too many layers relative to the size of the training dataset) is less robust, such as DNN-7; *ii)* convolutional layers: DNNs with large numbers of convolutional layers are less robust, e.g., compared with DNN-5, DNN-6 has an additional convolutional layer, but is significantly less robust; *iii)* batch-normalisation layers: adding a batch-normalisation layer may improve robustness, e.g., DNN-3 is more robust than DNN-2.

We remark that accuracy measured on a test dataset is not a good proxy for robustness: a DNN with higher accuracy is not necessarily more robust, e.g., DNN-1 and DNN-3 are more robust than DNN-6 and DNN-7, which have higher accuracy. DNNs may require a balance between robustness and their ability to generalise (proxied by testing accuracy) [Tsipras *et al.*, 2018]. DNN-4 is a good example, among our limited set.

Therefore, our tool DeepTRE can be used to perform model selection, given two neural networks with similar accuracy, or to find a model that offers a good balance between robustness and performance.

Table 1: Neuron coverage achieved by DeepTRE and other tools

	DeepTRE (%)	DeepConcolic (%)	DeepXplore (%)		
			light	occlusion	blackout
MNIST	**98.95**	97.60	80.77	82.68	81.61
CIFAR-10	**98.63**	84.98	77.56	81.48	83.25

5.5 Test Case Generation for DNNs

A variety of methods to automate testing of DNNs have been proposed recently [Pei *et al.*, 2017; Sun *et al.*, 2018]. The most widely used metric for the exhaustiveness of test suites for DNNs is *neuron coverage* [Pei *et al.*, 2017]. Neuron coverage quantifies the percentage of hidden neurons in the network that are activated at least once. We use ne to range over hidden neurons, and $V(ne, x)$ to denote the activation value of ne for test input x. Then $V(ne, x) > 0$ implies that ne is covered by the test input x.

The application of our algorithm to coverage-driven test case generation is straightforward; it only requires a minor modification to the optimisation problem in Definition 2. Given any neuron ne that is not activated by the test suite $\mathtt{T}_0$, we find the input with the smallest distance to any input in $\mathtt{T}_0$ that activates ne. We replace the constraint $cl(f, x_i) \neq cl(f, x_{0,i})$ in Equation 1 by

$$V(ne, x_i) \leq 0 \wedge V(ne, x_{0,i}) > 0. \tag{15}$$

The optimisation problem now searches for new inputs that activate the neuron ne, and the objective is to minimise the distance from the current set of test inputs $\mathtt{T}_0$.

We compare our tool DeepTRE with other state-of-the-art test case generation methods, including DeepConcolic [Sun *et al.*, 2018] and DeepXplore [Pei *et al.*, 2017]. All results are

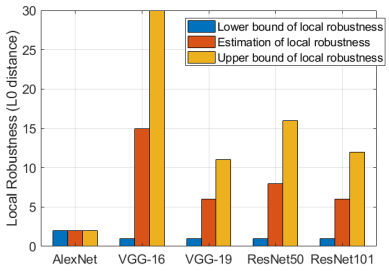


Figure 8: Lower bounds, upper bounds and estimates of local robustness for 5 ImageNet DNNs on a given image.

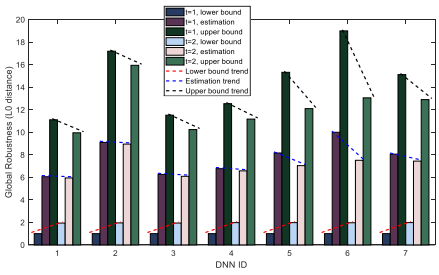


Figure 9: Lower bounds, upper bounds, and global robustness estimates for $t \in \{1, 2\}$ for seven DNN models.

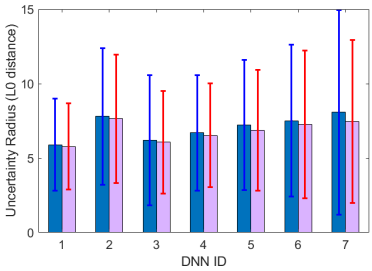


Figure 10: Means and standard deviations of uncertainty radius d_m for 1,000 test images at $t = 1, 2$.

Figure 11: Some adversarial examples found by our tool DeepTRE while generating test cases for high neuron coverage on MNIST and CIFAR-10 DNNs.

averaged over 10 runs or more. Table 1 gives the neuron coverage obtained by the three tools. We observe that DeepTRE yields much higher neuron coverage than both DeepConcolic and DeepXplore in any of its three modes of operation ('light', 'occlusion', and 'blackout'). Figure 11 depicts adversarial examples generated by our tool DeepTRE. We also observe that a significant proportion of the adversarial examples can be found using a relatively small L_0 distance. More experimental results can be found in Appendix H of [Ruan *et al.*, 2018b]. Overall, our tool DeepTRE offers an efficient approach to coverage-driven testing of DNNs.

6 Related Work

Global robustness evaluation is related to adversarial example generation. Most existing algorithms for this problem first compute a gradient (either a cost gradient or a forward gradient) and then perturb the input in different ways along the most promising direction on fooling the neural network. In view of the space limit, we omit the discussion of related work on adversarial attacks, and just mention that we have shown experimentally that our approach can obtain tighter upper bounds (i.e., smaller adversarial distances) at lower computational cost. We remark that we focus on the prevailing concept of robustness against small perturbations, called epsilon-adversarial examples in [Jacobsen *et al.*, 2019]; at present our method is unable to deal with invariance-based adversarial examples discussed there.

Robustness evaluation is usually performed w.r.t. a similarity metric on inputs, with L_p norms typically employed for this purpose. The L_0 metric has been used in [Papernot *et al.*, 2016; Carlini *et al.*, 2017; Huang *et al.*, 2017; Wicker *et al.*, 2018; Wu *et al.*, 2018] for adversarial attacks; our method is the first to provide anytime lower and upper bounds on robustness guarantees for the L_0 metric.

In addition to providing anytime upper/lower bounds on global robustness, our method also guarantees convergence. Existing approaches that offer guarantees instead focus on local (pointwise) robustness. These include reduction to constraint solving [Pulina and Tacchella, 2010; Katz *et al.*, 2017], abstract interpretation [Gehr *et al.*, 2018; Mirman *et al.*, 2018], exhaustive search [Huang *et al.*, 2017] or Monte Carlo tree search (MCTS) for Lipschitz networks [Wu *et al.*, 2018]. Another group of papers considers the problem of whether an output value of a DNN is reachable from a given input subspace [Lomuscio and Maganti, 2017; Ruan *et al.*, 2018a; Dutta *et al.*, 2017], by either reducing the problem to a MILP problem [Lomuscio and Maganti, 2017], or by considering the range of values [Dutta *et al.*, 2017], or by employing global optimisation [Ruan *et al.*, 2018a]. We also mention [Peck *et al.*, 2017], who compute a lower bound of local robustness for the L_2 norm. This is incomparable with our result because of the different distance metrics. In recent work [Gopinath *et al.*, 2018] the input vector space is partitioned using clustering and then the method of [Katz *et al.*, 2017] is used to check the individual partitions. However, none of the methods and tools above support the L_0 distance and provide *anytime* and *guaranteed* convergence to the true global robustness. Thus, our tool, DeepTRE, is complementary to existing approaches.

7 Conclusions

To the best of our knowledge, this is the *first* algorithm that evaluates global robustness of DNNs based on the Hamming distance with provable *guarantees*. We provide a tensor-based implementation of the technique to exploit the inherent parallelism. Our experimental results demonstrate wide applicability and efficiency of the method, with potential for real-time deployment. We hypothesise that the approach computes a good proxy for the robustness against physical attacks that rely on the manipulation of a small part of the objects that are to be recognised.

Acknowledgements

MK and WR have been partially supported by the EPSRC Programme Grant on Mobile Autonomy (EP/M019918/1). MW has been supported by the CSC-PAG Oxford Scholarship.

References

[Biggio *et al.*, 2013] Battista Biggio, Giorgio Fumera, and Fabio Roli. Security evaluation of pattern classifiers under attack. *IEEE Transactions on Knowledge and Data Engineering*, 26(4):984–996, 2013.

[Carlini and Wagner, 2017] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 39–57. IEEE, 2017.

[Carlini *et al.*, 2017] Nicholas Carlini, Guy Katz, Clark Barrett, and David L Dill. Ground-truth adversarial examples. *arXiv preprint arXiv:1709.10207*, 2017.

[Dutta *et al.*, 2017] Souradeep Dutta, Susmit Jha, Sriram Sanakaranarayanan, and Ashish Tiwari. Output range analysis for deep neural networks. *arXiv preprint arXiv:1709.09130*, 2017.

[Gehr *et al.*, 2018] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. AI^2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, 2018.

[Goodfellow *et al.*, 2015] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2015.

[Gopinath *et al.*, 2018] Divya Gopinath, Guy Katz, Corina S. Pasareanu, and Clark Barrett. DeepSafe: A data-driven approach for checking adversarial robustness of neural networks. In *Automated Technology for Verification and Analysis (ATVA)*, pages 3–19. Springer, 2018.

[Huang *et al.*, 2017] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *CAV*, pages 3–29. Springer, 2017.

[Jacobsen *et al.*, 2019] Jörn-Henrik Jacobsen, Jens Behrmann, Nicholas Carlini, Florian Tramèr, and Nicolas Papernot. Exploiting excessive invariance caused by norm-bounded adversarial robustness. *arXiv preprint arXiv:1903.10484*, 2019.

[Katz *et al.*, 2017] Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *CAV*. Springer, 2017.

[Kurakin *et al.*, 2016] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[Lomuscio and Maganti, 2017] Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward ReLU neural networks. *CoRR*, abs/1706.07351, 2017.

[Mirman *et al.*, 2018] Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pages 3575–3583, 2018.

[Narodytska *et al.*, 2017] Nina Narodytska, Shiva Prasad Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying properties of binarized deep neural networks. *CoRR*, abs/1709.06662, 2017.

[Papernot *et al.*, 2016] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *EuroS&P 2016*, 2016.

[Peck *et al.*, 2017] Jonathan Peck, Joris Roels, Bart Goossens, and Yvan Saeys. Lower bounds on the robustness to adversarial perturbations. In *Advances in Neural Information Processing Systems*, pages 804–813, 2017.

[Pei *et al.*, 2017] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. DeepXplore: Automated whitebox testing of deep learning systems. In *SOSP*, pages 1–18. ACM, 2017.

[Pulina and Tacchella, 2010] Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *CAV*, pages 243–257. Springer, 2010.

[Ruan *et al.*, 2018a] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. *The 27th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2651–2659, 2018.

[Ruan *et al.*, 2018b] Wenjie Ruan, Min Wu, Youcheng Sun, Xiaowei Huang, Daniel Kroening, and Marta Kwiatkowska. Global robustness evaluation of deep neural networks with provable guarantees for L_0 norm. *arXiv preprint arXiv:1804.05805*, 2018.

[Sun *et al.*, 2018] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. Concolic testing for deep neural networks. In *Automated Software Engineering (ASE)*, pages 109–119. ACM, 2018.

[Szegedy *et al.*, 2014] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.

[Tsipras *et al.*, 2018] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152*, May 2018.

[Wicker *et al.*, 2018] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. Feature-guided black-box safety testing of deep neural networks. In *TACAS*, LNCS, pages 408–426. Springer, 2018.

[Wu *et al.*, 2018] Min Wu, Matthew Wicker, Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. A game-based approximate verification of deep neural networks with provable guarantees. *To appear in Theoretical Computer Science. CoRR*, abs/1807.03571, 2018.