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

Wenjie Ruan¹, Min Wu¹, Youcheng Sun¹, Xiaowei Huang², Daniel Kroening¹ and Marta Kwiatkowska¹

¹Department of Computer Science, University of Oxford
²Department 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] [Limuscio 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 $L_{\infty}$ 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
Given a network $f$, we design a method that is well-suited for GPUs using tensor operations, 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 an intermediate bound and a new, homogeneous set whose distance to any adversarial example is possible for perturbation of up to $\epsilon$ 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 an intermediate bound and a new, homogeneous set whose distance to any adversarial example is possible for perturbation of up to $\epsilon$ 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 an intermediate bound and a new, homogeneous set whose distance to any adversarial example is possible for perturbation of up to $\epsilon$ 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 an intermediate bound and a new, homogeneous set whose distance to any adversarial example is possible for perturbation of up to $\epsilon$ 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].

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 $T_0$ of inputs, i.e., $f(T_0)$ and $cl(f(T_0))$, 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 $B(f, x_0, || \cdot ||_D, d) = \{x | ||x - x_0||_D \leq d\}$. 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 $T$, which is a set of i.i.d. inputs sampled from a distribution $\mu$ representing the problem $f$ is working on. We use $|T|$ to denote the number of inputs in $T$. When $|T| = 1$, we call it local robustness.

**Definition 2 (Robustness Evaluation)**

Given a network $f$, a finite set $T_0$ of inputs, and $|| \cdot ||_D$, robustness evaluation, denoted as $R(f; T_0, || \cdot ||_D)$, is an optimisation problem:

$$\min_{T} ||T - T_0||_D \text{ s.t. } cl(f, x_i) \neq cl(f, x_0)$$

for all $i \in \{1, \ldots, |T_0|\}$ where $T = (x_i)_{i=1,\ldots,|T_0|}$ and $T_0 = (x_0, i)_{i=1,\ldots,|T_0|}$.

Intuitively, we aim to find a minimum distance between the original set $T_0$ and a new, homogeneous set $T$ of inputs such that all inputs in $T_0$ 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. 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 $T_0$ (all inputs in $T_0$ are i.i.d.), we define

$$||T - T_0||_0 = \sum_{x \in T_0} ||x - x_0||_0$$

(2)

where $x \in T$ is a homogeneous input to $x_0 \in T_0$. While other norms such as $L_1$, $L_2$ and $L_\infty$ have been widely applied for generating adversarial examples [Papernot 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
Def. 3 (Sequences of Bounds) Given a robustness evaluation problem \( R(f, T_0, \| \cdot \|_D) \), a sequence \( L(T_0) = \{l_1, l_2, \ldots, l_k\} \subseteq \mathbb{R} \) is an incremental lower bound sequence if, for all \( 1 \leq i < j \leq k \), we have \( l_i \leq l_j \leq R(f, T_0, \| \cdot \|_D) \). The sequence is strict, denoted as \( L_s(T_0) \), if for all \( 1 \leq i < j \leq k \), we have either \( l_i < l_j \) or \( l_i = l_j \). Similarly, we can define a decremental upper bound sequence \( U(T_0) \) and a strict decremental upper bound sequence \( U_d(T_0) \).

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 \( t > 0 \),

\[
l_t \leq R(f, T_0, \| \cdot \|_D) \leq u_t
\]

holds.

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

\[
U_c(l_t, u_t) = \frac{1}{2}(l_t + u_t) \quad \text{and} \quad U_r(l_t, u_t) = \frac{1}{2}(u_t - l_t)
\]

The anytime evaluation of \( R(f, T_0, \| \cdot \|_D) \) at time \( t \), denoted as \( R_t(f, T_0, \| \cdot \|_D) \), 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 \( R(f, T_0, \| \cdot \|_D) \) 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.

Def. 5 (Complete Set of Subspaces for an Input) Given an input \( x_0 \in [0, 1]^n \) and a set of \( t \) dimensions \( T \subseteq \{1, \ldots, n\} \) 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 \( i \in \{1, \ldots, n\} \setminus T \). Furthermore, given an input \( x_0 \in [0, 1]^n \) and a number \( t \leq n \), we define

\[
X(x_0, t) = \{X_{x_0,T} \mid T \subseteq \{1, \ldots, n\}, |T| = t\}
\]

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, \( X(x_0, t) \) 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 \( T_0 \). Recall that \( f(x) = (c_1(x), \ldots, c_m(x)) \).

Def. 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

\[
S(X, x_0, j) = c_j(x_0) - \inf_{x \in X} c_j(x).
\]

Let \( t \) be an integer. We define the subspace sensitivity for \( T_0 \) and \( t \) as

\[
S(T_0, t) = (S(X_{x_0}, x_0, j_x))_{X_{x_0} \in X(x_0,t), x_0 \in T_0}
\]

where \( j_{x_0} = \arg \max_{c \in \{1, \ldots, m\}} c_i(x_0) \) 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 \( S(T_0, t) \) is the two-dimensional array of the maximal decreases of confidence values of the classification labels for all subspaces in \( X(x_0,t) \) and all inputs in \( T_0 \). It is not hard to see that \( S(X, x_0, j) \geq 0 \).

Given a test dataset \( T_0 \) and an integer \( t > 0 \), the number of elements in \( S(T_0, t) \) is in \( O(T_0 \cdot n^t) \), i.e., polynomial in \( |T_0| \) and exponential in \( t \). Note that, by Equation (6), every element in \( S(T_0, t) \) represents an optimisation problem. That is, for \( T_0 \), 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 \( T \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N} \) 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.

Def. 7 (Mode-n Unfolding and Folding) Given a tensor \( T \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N} \), the \( n \)-mode unfolding of \( T \) is a matrix \( U_n(T) \in \mathbb{R}^{I_n \times I^M} \) such that \( M = \prod_{k=1,k\neq n}^{N} I_k \) and \( U_n(T) \) is defined by the mapping from element \((i_1, \ldots, i_N)\) to \((i_n, j)\), with

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

Similarly, the tensor folding \( F \) 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 $T_0$, each $x_i \in T_0$ generates a complete set $X(x_i, t)$ of subspaces. Let $|T_0| = p$ and $|X(x_i, t)| = k$. Note that, for different $x_i$ and $x_j$, we have $|X(x_i, t)| = |X(x_j, t)|$. 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 $X_t \in X(x_i, t)$ into a two-dimensional grid $G(X_t) \in \mathbb{R}^{n \times \Delta^t}$. We thus can formulate the following tensor:

$$T(T_0, t) = \text{Tensor}(G(X_t))_{x_i \in T_0, x_i \in X(x_i, t)} \in \mathbb{R}^{n \times \Delta^t \times p \times k}$$  \hspace{1cm} (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 $T_{[1]}(T(T_0, t)) \in \mathbb{R}^{n \times M}$ such that $M = \Delta^t \cdot p \cdot k$. This tensor can be fed into the DNN $f$ to obtain

$$Y(T_0, t) = f(T_{[1]}(T(T_0, t))) \in \mathbb{R}^M. \hspace{1cm} (10)$$

After computing $Y(T_0, t)$, we apply a tensor folding operation to obtain

$$Y(T_0, t) = F(Y(T_0, t)) \in \mathbb{R}^{\Delta^t \times p \times k}. \hspace{1cm} (11)$$

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 $Y(T_0, t)$, we search the minimum values along the first dimension to obtain:

$$V(T_0, t)_{\min} = \min(Y(T_0, t), 1) \in \mathbb{R}^{p \times k}. \hspace{1cm} (12)$$

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

$$V(T_0, t) = (c_{jx_i}(x_i), ..., c_{jx_i}(x_i))_{x_i \in T_0} \in \mathbb{R}^{p \times k} \hspace{1cm} (13)$$

from the set $T_0$. Recall that $j_{x_i} = \arg \max_{x \in \{1, ..., m\}} c_k(x_i)$. Intuitively, $V(T_0, t)$ is the tensor that contains the starting points of the optimisation problems and $V(T_0, t)_{\min}$ the resulting optimal values. The following theorem shows the correctness of our computation, where $S(T_0, t)$ has been defined in Definition 2.

**Theorem 1.** Let $T_0$ be a test dataset and $t$ an integer. We have $S(T_0, t) = V(T_0, t) - V(T_0, t)_{\min}$.

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 $S(T_0, t) \in \mathbb{R}^{n \times p \times k}$ be the tensor obtained by replacing every element in $S(T_0, t)$ by its corresponding input that, according to the computation of $V(T_0, t)_{\min}$, causes the largest decreases of the confidence values of the classification labels. We call $S(T_0, t)$ the **solution tensor** of $S(T_0, t)$. The computation of $S(T_0, t)$ can be done using very few tensor operations over $T(T_0, t)$ and $Y(T_0, t)$, which have been given in Section 4.1. We omit the details.

---

1. 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 $S(T_0, t)$ and $S(T_0, t)$ w.r.t. decreasing values in $S(T_0, t)$. Then, we retrieve the first row of the third dimension in tensor $S(T_0, t)$, i.e., $S(T_0, t)[i; ; ] \in \mathbb{R}^{n \times p}$, and check whether $c(f, S(T_0, t)[i; ; ]) = c(f, T_0)$. The result is an array of Boolean values, each of which is associated with an input $x_i \in T_0$. If any element associated with $x_i$ in the resulting array is false, we conclude that $d_m(f, x_i, \| \cdot \|_p) = t - 1$, 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 $S(T_0, t)$, 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 $S(T_0, t)$ for every input in $T_0$, 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 $T$.

- **Initialisation:** $\mathcal{N}[i; i; 1] = S(T_0, t)[i; ; 1]$. 

- **Iteratively construct the $i$-th row until $i = k$:**

$$\mathcal{N}[i; i; i] = \begin{cases} \mathcal{N}[i; i; i-1] \oplus \mathcal{N}[i; i-1] \oplus S(T_0, t)[i; ; i] \\ \mathcal{N}[i; i; i-1] \oplus S(T_0, t)[i; ; i] \end{cases}$$

where $\oplus$, $\ominus$, and $\otimes$ are tensor operations: $\mathcal{N}_1 \oplus \mathcal{N}_2$ removes the corresponding non-zero elements in $\mathcal{N}_2$ from $\mathcal{N}_1$; further, $\mathcal{N}_1 \ominus \mathcal{N}_2$ retains those elements that have the same values and sets the other elements to 0; finally, $\mathcal{N}_1 \otimes \mathcal{N}_2$ merges the non-zero elements from two tensors. The two operands of these operations are required to have the same type. Intuitively, $\mathcal{N}[i; i; i]$ represents the result of applying the first $i$ perturbations recorded in $S(T_0, t)[i; ; i]$, which contains the perturbations up to index $i - 1$ (i.e., $\mathcal{N}[i; i; i-1]$) plus the new perturbation recorded in $\mathcal{N}[i; i; i]$.

Subsequently, we unfold $\mathcal{N}$ and pass the result to the DNN $f$, which yields the classification labels $Y(U_{[1]}(\mathcal{N})) \in \{1, ..., m\}^{p \times k}$. After that, a tensor folding operation is applied to obtain $Y(U_{[1]}(\mathcal{N})) \in \{1, ..., m\}^{p \times k \times p}$. 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

$$T = \{\mathcal{N}[i; i; m] \in \mathbb{R}^{n \times p} \mid x_i \in T_0\}, \hspace{1cm} (14)$$

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

After computing $S(T_0, t)$, we only need one further DNN query to obtain all upper bounds for a given test dataset $T_0$.

**Tightening the Upper Bounds**

There may be redundancies in $T - T_0$, i.e., not all the changes in $T - T_0$ are necessary to observe a misclassification. We therefore remove the redundancies and thereby tighten the
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 \( \mathcal{T} \) and \( \mathcal{T}_0 \), respectively, for \( i \in \{1, \ldots, |\mathcal{T}_0|\} \). 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 \( l_{0,i} = x_i \oplus x_{0,i} \) be the part of \( x_{0,i} \) on which \( x_{0,i} \) and \( x_i \) are the same. Therefore, \( x_{0,i} = z_{0,i} \oplus l_{0,i} \).

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

\[
\min_{l_i} ||L_0 - L_i||_D \quad \text{s.t.} \quad \text{cl}(f, z_{0,i} \oplus l_{1,i}) \neq \text{cl}(f, z_{0,i} \oplus l_{0,i})
\]

where \( i \in \{1, \ldots, |\mathcal{T}_0|\}, \mathcal{L}_0 = \{l_{0,i}\}_{i=1}^{\mathcal{T}_0}, \mathcal{L}_1 = \{l_{1,i}\}_{i=1}^{\mathcal{T}_0}, \text{and} \ l_{0,i}, l_{1,i} \in \{0, 1\}^{\mathcal{T}} \). 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 \( \oplus \) 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 utility 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 \( \times \) 28 and sDNN on images resized to \( 14 \times 14 \). 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 \( |\mathcal{T}| = 1 \). 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 \( U_r(l_t, u_t) = 1.97 \) 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 5(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.
We train MNIST and CIFAR-10 DNNs and compare with

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. 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.

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].

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.

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

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

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.
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 2 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.

5.5 Test Case Generation for DNNs

A variety of methods to automate testing of DNNs have been proposed recently [Pei et al., 2017] and DeepXplore [Pei 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 \(T_0\), we find the input with the smallest distance to any input in \(T_0\) that activates \(ne\). We replace the constraint \(cl(f, x, i) \neq cl(f, x, 0)\) in Equation 1 by

\[
V(ne, x) \leq 0 \land V(ne, x, 0) > 0. \tag{15}
\]

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 \(T_0\).

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
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.