SOUND AND COMPLETE NEURAL NETWORK REPAIR WITH MINIMALITY AND LOCALITY GUARANTEES

Feisi Fu  
Division of Systems Engineering  
Boston University  
Boston, MA 02215  
fufeisi@bu.edu

Wenchao Li  
Department of Electrical and Computer Engineering  
Boston University  
Boston, MA 02215  
wenchao@bu.edu

ABSTRACT

We present a novel methodology for repairing neural networks that use ReLU activation functions. Unlike existing methods that rely on modifying the weights of a neural network which can induce a global change in the function space, our approach applies only a localized change in the function space while still guaranteeing the removal of the buggy behavior. By leveraging the piecewise linear nature of ReLU networks, our approach can efficiently construct a patch network tailored to the linear region where the buggy input resides, which when combined with the original network, provably corrects the behavior on the buggy input. Our method is both sound and complete – the repaired network is guaranteed to fix the buggy input, and a patch is guaranteed to be found for any buggy input. Moreover, our approach preserves the continuous piecewise linear nature of ReLU networks, automatically generalizes the repair to all the points including other undetected buggy inputs inside the repair region, is minimal in terms of changes in the function space, and guarantees that outputs on inputs away from the repair region are unaltered. On several benchmarks, we show that our approach significantly outperforms existing methods in terms of locality and limiting negative side effects. Our code is available on GitHub: https://github.com/BU-DEPEND-Lab/REASSURE.

1 Introduction

Deep neural networks (DNNs) have demonstrated impressive performances on a wide variety of applications ranging from transportation Bojarski et al. [2016] to health care Shahid et al. [2019]. However, DNNs are not perfect. In many cases, especially when the DNNs are used in safety-critical contexts, it is important to correct erroneous outputs of a DNN as they are discovered after training. For instance, a neural network in charge of giving control advisories to the pilots in an aircraft collision avoidance system, such as the ACAS Xu network from Julian et al. [2019], may produce an incorrect advisory for certain situations and cause the aircraft to turn towards the incoming aircraft, thereby jeopardizing the safety of both airplanes. In this paper, we consider the problem of neural network repair, i.e. given a trained neural network and a set of buggy inputs (inputs on which the neural network produces incorrect predictions), repair the network so that the resulting network on those buggy inputs behave according to some given correctness specification. Ideally, the changes to the neural network function should be small so that the outputs on other inputs are either unchanged or altered in a small way. Existing efforts on neural network repair roughly fall into the following three categories.

1. Retraining/fine-tuning. The idea is to retrain or fine-tune the network with the newly identified buggy inputs and the corresponding corrected outputs. Methods include counterexample-guided data augmentation Dreossi et al. [2018], Ren et al. [2020], editable training Sinitsin et al. [2020] and training input selection Ma et al. [2018]. One major weakness of these approaches is the lack of formal guarantees – at the end of retraining/fine-tuning, there is no guarantee that the given buggy inputs are fixed and no new bugs are introduced. In addition, retraining can be very expensive and requires access to the original training data which is impractical in cases where the neural network is obtained from a third party or the training data is private. Fine-tuning, on the other hand, often faces the issue of catastrophic forgetting Kirkpatrick et al. [2017].
2. Direct weight modification. These approaches directly manipulate the weights in a neural network to fix the buggy inputs. The repair problem is typically cast into an optimization problem or a verification problem. For example, Dong et al. [2020] proposes to minimize a loss defined based on the buggy inputs. Goldberger et al. [2020] uses an SMT solver to identify minimal weight changes to the output layer of the network so that the undesirable behaviors are removed. In general, the optimization-based approach cannot guarantee removal of the buggy behaviors, and the verification-based approach does not scale beyond networks of a few hundred neurons. In addition, both approaches suffer from substantial accuracy drops on normal inputs since weight changes may be a poor proxy for changes in the function space.

3. Architecture extension. The third category of approaches extends the given NN architecture, such as by introducing more weight parameters, to facilitate more efficient repairs. The so-called Decoupled DNN architecture Sotoudeh and Thakur [2021] is the only work we know that falls into this category. Their idea is to decouple the activations of the network from values of the network by augmenting the original network. Their construction allows the formulation of any single-layer repair as an linear programming (LP) problem. The decoupling, however, causes the repaired network to become discontinuous (in the functional sense). In addition, it still cannot isolate the output change to a single buggy input from the rest of the inputs.

In addition to the aforementioned limitations, a common weakness that is shared amongst these methods is that the induced changes, as a result of either retraining or direct weight modification, are global. This means that a correct behavior on another input, regardless of how far it is away from the buggy input, may not be preserved by the repair. Worse still, the repair on a new buggy input can end up invalidating the repair on a previous buggy input. The fundamental issue here is that limiting the changes to a few weights or a single layer only poses a structural constraint (often for ease of computation); it does not limit the changes on the input-output mapping of the neural network. It is known that even a single weight change can have a global effect on the output of a neural network.

In this paper, we propose REASSURE, a novel methodology for neural network repair with locality, minimality, soundness and completeness guarantees. Our methodology targets continuous piecewise linear (CPWL) neural networks, specifically those that use the ReLU activation functions. The key idea of our approach is to leverage the CPWL property of ReLU networks to synthesize a patch network tailored to the linear region where the buggy input resides, which when combined with the original network, provably corrects the behavior on the buggy input. Our approach is both sound and complete – the repaired network is guaranteed to fix the buggy input, and a patch is guaranteed to be found for any buggy input. Moreover, our approach preserves the CPWL nature of ReLU networks, automatically generalizes the repair to all the points including other undetected buggy inputs inside the repair region, is minimal in terms of changes in the function space, and guarantees that outputs on inputs away from the repair region are unaltered. Figure 1 provides an illustrative comparison of our approach with other methods. Table 1 compares our approach with representative related works in terms of theoretical guarantees. We summarize our contributions below.

- We present REASSURE, the first sound and complete repair methodology for ReLU networks with strong theoretical guarantees.
- Our technique synthesizes a patch network, which when combined with the original neural network, provably corrects the behavior on the buggy input. This approach is a significant departure from existing methods that rely on retraining or direct weight manipulation.
- Our technique is both effective and efficient – across a set of benchmarks, REASSURE can efficiently correct a set of buggy inputs or buggy areas with little or no change to the accuracy and overall functionality of the network.
Table 1: Comparing REASSURE with representative related works in terms of theoretical guarantees. CPWL stands for continuous piecewise linearity. Area repair means repairing all the (infinitely many) points inside an area. Limited side effect means the repair can limit potential adverse effects on other inputs. MDNN is the verification-based approach from Goldberger et al. [2020]. PRDNN is the Decoupled DNN approach from Sotoudeh and Thakur [2021]. REASSURE is the only method that can provide all the guarantees.

2 Background

2.1 Deep Neural Networks

An $R$-layer feed-forward DNN $f = \kappa_R \circ \sigma \circ \kappa_{R-1} \circ \ldots \circ \sigma \circ \kappa_1 : X \to Y$ is a composition of linear functions $\kappa_r, r = 1, 2, \ldots, R$ and activation function $\sigma$, where $X \subseteq \mathbb{R}^m$ is a bounded input domain and $Y \subseteq \mathbb{R}^n$ is the output domain. Weights and biases of linear function $\{\kappa_r\}_{r=1,2,...,R}$ are parameters of the DNN.

We call the first $R - 1$ layers hidden layers and the $R$-th layer the output layer. We use $z_j^i$ to denote the $i$-th neuron (before activation) in the $j$-th hidden layer.

In this paper, we focus on ReLU DNNs, i.e. DNNs that use only the ReLU activation functions. It is known that an $\mathbb{R}^m \to \mathbb{R}$ function is representable by a ReLU DNN if and only if it is a continuous piecewise linear (CPWL) function Arora et al. [2016]. The ReLU function is defined as $\sigma(x) = \max(x, 0)$. We say that $\sigma(x)$ is activated if $\sigma(x) = x$.

2.2 Linear Regions

A linear region $\mathcal{A}$ is the set of inputs that correspond to the same activation pattern in a ReLU DNN $f$ Serra et al. [2017]. Geometrically, this corresponds to a convex polytope, which is an intersection of half spaces, in the input space $X$ on which $f$ is linear. We use $f \rvert_\mathcal{A}$ to denote the part of $f$ on $\mathcal{A}$.

2.3 Correctness Specification

A correctness specification $\Phi = (\Phi_{in}, \Phi_{out})$ is a tuple of two polytopes, where $\Phi_{in}$ is the union of some linear regions and $\Phi_{out}$ is a convex polytope. A DNN $f$ is said to meet a specification $\Phi = (\Phi_{in}, \Phi_{out})$, denoted as $f \models \Phi$, if and only if $\forall x \in \Phi_{in}, f(x) \in \Phi_{out}$.

Example 1. For a classification problem, we can formally write the specification that “the prediction of any point in an area $\mathcal{A}$ is class $k$” as $\Phi = (\Phi_{in}, \Phi_{out})$, where $\Phi_{in} = \mathcal{A}$ and $\Phi_{out} = \{y \in \mathbb{R}^n | y_k \geq y_i, \forall i \neq k\}$.

2.4 Problem Definition

In this paper, we consider the following two repair problems.

Definition 1 (Area repair). Given a correctness specification $\Phi = (\Phi_{in}, \Phi_{out})$ and a ReLU DNN $f \not\models \Phi$, the area repair problem is to find a modified ReLU DNN $\hat{f}$ such that $\hat{f} \models \Phi$.

Note that we do not require $\hat{f}$ to have the same structure or parameters as $f$ in this definition. If $\Phi_{in}$ contains a single (buggy) linear region, we refer to this as single-region repair. If $\Phi_{in}$ contains multiple (buggy) linear regions, we refer to it as multi-region repair.

Definition 2 (Point-wise repair). Given a set of buggy inputs $\{\bar{x}_1, \ldots, \bar{x}_L\} \subseteq \Phi_{in}$ with their corresponding correct outputs $\{y_1, \ldots, y_L\}$ and a ReLU DNN $f$, the point-wise repair problem is to find a modified ReLU DNN $\hat{f}$ such that $\forall i, \hat{f}(\bar{x}_i) = y_i$.

\[1\text{Note that here } y \text{ is the output of the layer right before the softmax layer in a classification network.}\]
We call the minimal variants of area repair and point-wise repair minimal area repair and minimal point-wise repair respectively. Minimality here is defined with respect to the maximum distance between \( f \) and \( \hat{f} \) over the whole input domain \( X \). A point-wise repair can be generalized to an area repair through the following result.

### 2.5 From Buggy Inputs to Buggy Linear Regions

The linear region where an input \( x \) resides can be computed as follows.

**Lemma 1.** [Lee et al. 2019] Consider a ReLU DNN \( f \) and an input \( x \in X \). For every neuron \( z_j^i \), it induces a feasible set

\[
A_j^i(x) = \begin{cases} 
\{ \bar{x} \in X | (\nabla_x z_j^i)^T \bar{x} + z_j^i \geq 0 \} & \text{if } z_j^i \geq 0 \\
\{ \bar{x} \in X | (\nabla_x z_j^i)^T \bar{x} + z_j^i \leq 0 \} & \text{if } z_j^i < 0 
\end{cases}
\]

The set \( A(x) = \cap_{i,j} A_j^i(x) \) is the linear region that includes \( x \). Note that \( A(x) \) is essentially the \( H \)-representation of the corresponding convex polytope.

### 2.6 Repair Desiderata

We argue that an effective repair algorithm for ReLU DNN should satisfy the following criteria.

- **Preservation of CPWL:** Given that the original network \( f \) models a CPWL function, the repaired network \( \hat{f} \) should still model a CPWL function.

- **Soundness:** A sound repair should completely remove the known buggy behaviors, i.e. it is a solution to the point-wise repair problem defined in Definition 2.

- **Completeness:** Ideally, the algorithm should always be able to find a repair for any given buggy input if it exists.

- **Generalization:** If there exists another buggy input \( \bar{x}' \) in the neighborhood of \( \bar{x} \) (e.g. the same linear region), then the repair should also fix it. For example, suppose we have an \( \bar{x} \) that violates a specification which requires the output to be within some range. It is almost guaranteed that there exists another (and infinitely many) \( \bar{x}' \) in the same linear region that also violates the specification.

- **Locality:** We argue that a good repair should only induce a localized change to \( f \) in the function space. For example, in the context of ReLU DNN, if a linear region \( B \) does not border the repair region \( A \), i.e. \( B \cap A = \emptyset \), then \( \hat{f}|_B(x) = f|_B(x) \).

- **Minimality:** Some notion of distance between \( f \) and \( \hat{f} \) such as \( \max |f - \hat{f}| \) should be minimized. Note that this is a significant departure from existing methods that focus on minimizing the change in weights which has no guarantee on the amount of change in the function space.

- **Limited side effect:** Repairing a buggy point should not adversely affect points that were originally correct. For example, repairing a buggy input \( \bar{x} \) in region \( A \) should not change another region from correct to incorrect. Formally, for any linear region \( C \) who is a neighbor of \( A \), i.e. \( C \cap A \neq \emptyset \), if \( f|_C \models \Phi \), then \( \hat{f}|_C \models \Phi \).

- **Efficiency:** The repair algorithm should terminate in polynomial time with respect to the size of the neural network and the number of buggy inputs.

### 3 Our Approach

We will first describe our approach to single-region repair and then present our approach to multi-region repair which builds on results obtained from the single-region case.

Given a linear region \( A \), our overarching approach is to synthesize a patch network \( h_A \) such that \( \hat{f} = f + h_A \) and \( \hat{f} \models \Phi \). The patch network \( h_A \) is a combination of two sub-networks: a support network \( g_A \) which behaves like a characteristic function to ensure that \( h_A \) is almost only active on \( A \), and an affine patch function network \( p_A(x) = cx + d \) such that \( (f + p_A) \models \Phi \) on \( A \).

#### 3.1 Running Example

We use the following example to illustrate our idea.
We will first consider finding appropriate \( p \) where \( A \) violates the criterion of preservation of CPWL. In Theorem 3, we will prove that we can still guarantee limited side effects on the whole input domain outside of \( A \). The design of the patch network \( h_A \) ensures locality for the final patch.

**Example 2.** Consider repairing the ReLU DNN \( f \) in Figure 2 according to the correctness specification \( \Phi : \forall x \in [0,1]^2, y \in [0,2] \). The DNN consists of a single hidden layer with two neurons \( z_1 \) and \( z_2 \), where \( y = \sigma(z_1) + \sigma(z_2) \). \( z_1 = x_1 + 2x_2 - 1 \) and \( z_2 = 2x_1 - x_2 \).

The only linear region that violates our specification is \( A = \{ x \mid 1 \geq x_1, 1 \geq x_2, x_1 + 2x_2 - 1 \geq 0, 2x_1 - x_2 \geq 0 \} \). (\( \bar{x} = (0.9, 0.9) \in [0,1]^2 \) but \( f(\bar{x}) = 2.6 \notin [0,2] \))

The network \( f(x) \) on the linear region \( A \) is the affine function \( f|_A(x) = 3x_1 + x_2 - 1 \). Our algorithm first sets up an affine function \( p_A(x) \) that minimally repairs \( f \) on \( A \), such that \( \forall x \in A, f(x) + p_A(x) \in [0,2] \). Later in the paper, we will show \( p_A(x) \) can be found by solving a LP problem. The resulting patch function is \( p_A(x) = -\frac{1}{2}x_1 - \frac{1}{2}x_2 \).

However, directly apply \( f(x) + p_A(x) \) as the patch network will have side effects on areas outside of \( A \). Our strategy is to combine \( p_A(x) \) with a support network \( g_A(x) \) which outputs 1 on \( A \) and drops to 0 quickly outside of \( A \). The final repaired network is \( f(x) + \sigma(p_A(x) + g_A(x, 10) - 1) - \sigma(-p_A(x) + g_A(x, 10) - 1) \). This structure makes \( p_A \) almost only active on \( A \) and achieve a localized repair. Observe that this is still a ReLU DNN.

### 3.2 Support Networks

Support networks are neural networks with a special structure that can approximate the characteristic function of a convex polytope. They are keys to ensuring localized repairs in our algorithm.

Assume that the linear region we need to repair is \( A = \{ x \mid a_i x \leq b_i, i \in I \} \), where \(|I|\) is the number of linear inequalities. The support network of \( A \) is defined as:

\[
g_A(x, \gamma) = \sigma(\sum_{i \in I} g(b_i - a_i x, \gamma) - |I| + 1)
\]

(2)

where \( g(x, \gamma) = \sigma(\gamma x + 1) - \sigma(\gamma x) \) and \( \gamma \in \mathbb{R} \) is a parameter of our algorithm that controls the slope of support networks.

**Remark:** For any \( x \in A \), we have \( g_A(x, \gamma) = 1 \), i.e. the support network is fully activated. For any \( x \notin A \), if for one of \( i \in I \), we have \( a_i x - b_i \leq -1/\gamma \), then \( g_A(x, \gamma) = 0 \).

Observe that \( g_A(x, \gamma) \) cannot be zero if \( x \) is very close to \( A \), otherwise the resulting function will be discontinuous and violates the criterion of preservation of CPWL. In Theorem 3, we will prove that we can still guarantee limited side effects on the whole input domain outside of \( A \).

### 3.3 Affine Patch Functions

We consider an affine patch function \( p_A(x) = c x + d \), where matrix \( c \) and vector \( d \) are undetermined coefficients. In a later section, the design of patch network will ensure that on the patch area \( A \), the repaired network is \( f(x) + p_A(x) \).

We will first consider finding appropriate \( c \) and \( d \) such that \( f(x) + p_A(x) \) satisfy the specification on \( A \).
To satisfy the specification $\Phi$, we need $f(x) + p_A(x) \in \Phi_{out}$ for all $x \in A$. To obtain a minimal repair, we minimize $\max_{x \in A} |p_A(x)|$. Thus, we can formulate the following optimization problem

$$\begin{align*}
\min_{c,d} & \max_{x \in A} |p_A(x)| = |cx + d| \\
(\text{subject to}) & (c,d) \in \{(c,d) \mid f(x) + cx + d \in \Phi_{out}, \forall x \in A\}
\end{align*}$$

(3)

Notice that this is not a LP since both $c$ and $x$ are variables and we have a $cx$ term in the objective.

In general, one can solve it by enumerating all the vertices of $A$. Since $\Phi_{out}$ is a convex polytope, we have

$$f(x) + p_A(x) \in \Phi_{out} \text{ for all } x \in A \iff f(v_s) + p_A(v_s) \in \Phi_{out} \text{ for } s = 1, 2, ..., S.$$  

(4)

and

$$\max_{x \in A} |cx + d| = \max_{s=1,2,...,S} |cv_s + d|$$

(5)

Hence, we can solve the following equivalent LP.

$$\begin{align*}
\min_{c,d} H & \\
H & \geq (cv_s + d) \text{, where } H_1 \text{ is the maximum value of the following inner LP} \\
H & \geq - (cv_s + d) \text{, for } s = 1, 2, ..., S \text{ and } i = 1, 2, ..., m \\
f(v_s) + p_A(v_s) & \in \Phi_{out} \text{, for } s = 1, 2, ..., S \\

(6)
\end{align*}$$

where $H \in \mathbb{R}$ and will take $\max_{s=1,2,...,S} |cv_s + d|$ when optimal.

### 3.4 Repair via Robust Optimization

In general, the number of vertices of a convex polytope can be exponential in the size of its H-representation [Henk et al. 1997] and enumerating the vertices of a convex polytope is known to be expensive especially when the input dimension is large [Bremner 1997]. In this section, we show that we can convert problem (3) to an LP via robust optimization [Ben-Tal et al. 2009] without vertex enumeration and make our algorithm much more efficient.

The optimization problem (3) can be converted to the following optimization problem, assuming $\Phi_{out} = \{y | a_{out} \cdot y \leq b_{out}\}$:

$$\begin{align*}
\min_{c,d} H & \\
H & \geq H_1, \text{ where } H_1 \text{ is the maximum value of the following inner LP} \\
H & \geq H_2, \text{ where } H_2 \text{ is the maximum value of the following inner LP} \\
b_{out} & \geq H_3, \text{ where } H_3 \text{ is the maximum value of the following inner LP}
\end{align*}$$

(7)

For the inner LPs, since we only care about the maximum value and not the feasible solution that reaches the maximum, we can take the dual of inner LPs to avoid enumerating the vertices of $A$. Take the first inner LP as an example:

$$H_1 = \max_{x \in A} cx + d = d + \max_{ax \leq b} cx \text{ dual} = d + \min_{p} p\cdot b$$

$$\begin{align*}
& a'p = c \\
& p \geq 0
\end{align*}$$

(8)

Therefore, we have the following equivalent LP for optimization problem (3). Taking the dual of the inner LP to convert the whole problem to an LP is known as taking the robust counterpart [Ben-Tal et al. 2009] of the original problem.

$$\begin{align*}
\min_{c,d,p_1,p_2,q,H} & H \\
H & \geq p_1'b + d, a'p_1 = c, p_1 \geq 0 \\
H & \geq p_2'b - d, a'p_2 = -c, p_2 \geq 0 \\
b_{out} & \geq q'b + a_{out}(d_f + d), a'q = a_{out}(c_f + c), q \geq 0
\end{align*}$$

(9)

where $f(x) = c_f x + d_f$. 

6
With a support network will invalidate both repairs and cannot guarantee that the repaired DNN will meet the specification where $K$ goes to zero quickly if $\gamma R$ goes to zero quickly if $\gamma$ is large. When $g_A(x, \gamma) = 0$, we have $h_A(x, \gamma) = \sigma(p_A(x) - K) - \sigma(-p_A(x) - K) = 0$.

The repaired network $\hat{f}(x) = f(x) + h_A(x, \gamma)$. Since $f$ and $h_A$ are both ReLU DNNs, we have $\hat{f}$ is also a ReLU DNN. We will give the formal guarantees on correctness in Theorem 1.

### 3.5 Single-Region Repairs

With a support network $g_A$ and an affine patch function $p_A$, we can synthesize the final patch network as follows:

$$h_A(x, \gamma) = \sigma(p_A(x) + K \cdot g_A(x, \gamma) - K) - \sigma(-p_A(x) + K \cdot g_A(x, \gamma) - K)$$

where $K$ is a vector with every entry is equal to the upper bound of $\{|p_A(x)| + \infty | x \in X\}$.

**Remark:** For $x \in A$, $g_A(x, \gamma) = 1$, then we have $h_A(x, \gamma) = \sigma(p_A(x)) - \sigma(-p_A(x)) = p_A(x)$. For $x \notin A$, $g_A(x, \gamma)$ goes to zero quickly if $\gamma$ is large.

### 3.6 Multi-Region Repairs

Suppose there are two linear regions, $A_1$ and $A_2$, that need to be repaired, and we have generated the affine patch function $p_{A_1}(x)$ for $A_1$ and $p_{A_2}(x)$ for $A_2$.

If $A_1 \cap A_2 = \emptyset$, then we can repair $f(x)$ with $\hat{f}(x) = f(x) + h_{A_1}(x, \gamma) + h_{A_2}(x, \gamma)$ directly, since $h_{A_1}(x, \gamma)$ and $h_{A_2}(x, \gamma)$ will not be nonzero at the same time when $\gamma$ is large enough.

However, if $A_1 \cap A_2 \neq \emptyset$, for any $x \in A_1 \cap A_2$, both $h_{A_1}(x, \gamma)$ and $h_{A_2}(x, \gamma)$ will alter the value of $f$ on $x$, which will invalidate both repairs and cannot guarantee that the repaired DNN will meet the specification $\Phi$. To avoid such over-repairs, our strategy is to first repair $A_1 \cup A_2$ with $p_{A_1}(x)$, and then repair $A_2$ with $p_{A_2}(x) - p_{A_1}(x)$. Figure 3 provides an illustration of a three-region case.

In general, for multi-region repair, we note $\{A_l\}_{l=1,2,\ldots,L}$ are all the buggy linear regions. Then we compute the support network $g_{A_l}(x, \gamma)$ and affine patch function $p_{A_l}(x)$ for each $A_l$. Note that this computation can be done in parallel.

Once we have $g_{A_l}(x, \gamma)$ and $p_{A_l}(x)$, we can “stitch” multiple local patches into a final patch as follows.

$$h(x, \gamma) = \sum_{l} [\sigma(p_{A_l}(x) - p_{A_{l-1}}(x) + \max_{j \geq l} \{g_{A_j}(x, \gamma)\} K_l - K_l) - \sigma(-p_{A_l}(x) + p_{A_{l-1}}(x) + \max_{j \geq l} \{g_{A_j}(x, \gamma)\} K_l - K_l)]$$

where $K_l$ is the upper bound of $\{|p_{A_l}(x) - p_{A_{l-1}}(x)|, x \in X\}$ and $p_{A_{0}}(x) = 0$.

**Remark:** $\max_{j \geq l} \{g_{A_j}(x, \gamma)\}$ is a support function for $\cup_{j \geq l} A_j$ and its value is 1 for any $x \in \cup_{j \geq l} A_j$.

### 3.7 Feature-Space Repairs

In general, when repairing a large DNN with a high input dimension, the number of linear constraints for one patch area $A$ will be huge and pose a challenge to solving the resulting LP. One advantage of our approach, which can be used to mitigate this problem, is that it allows for point-wise and area repairs in the feature space in a principled manner, i.e. constructing a patch network starting from an intermediate layer. This approach still preserves soundness and completeness, and is fundamentally different from arbitrarily picking a single layer for repair.

Specifically, for an $R$-layer DNN $f$, we split $f$ into two networks $f_1$ and $f_2$ according to a hidden layer, say the $j_{th}$ hidden layer, where $f_1$ is the function of the first $j$ layers, $f_2$ is the function of the last $R - j$ layers, and $f = f_2 \circ f_1$. The output space of $f_1$ is a feature space. For any buggy input $\tilde{x}$, $f_1(\tilde{x})$ is the corresponding buggy feature. The
point-wise repair problem in a feature space is to repair the behavior of \( f_2 \) on buggy features \( \{ f_1(\bar{x}_1), \ldots, f_1(\bar{x}_L) \} \). Note that this will automatically repair the behavior of \( f \) on buggy points \( \{ \bar{x}_1, \ldots, \bar{x}_L \} \).

Repairing in a feature space has the benefit of making the repair process more computation-friendly and reducing the parameter overhead of the additional networks, and has the potential to generalize the repair to undetected buggy inputs with similar features. It loses the locality guarantee in the input space but still preserves locality in the feature space.

4 Theoretical Guarantees

In this section, we present the theoretical guarantees that REASSURE provide, and point the readers to proofs of the theorems in the Appendix.

4.1 Soundness & Completeness

Theorem 1 (Soundness). The repaired DNN \( \hat{f} \) returned by REASSURE is guaranteed to satisfy the specification \( \Phi \).

Theorem 2 (Completeness). REASSURE can always find a solution to the minimal point-wise repair or the minimal area repair problem.

4.2 Limited Side Effect of Patch Networks

For any \( \mathcal{A} \), the support network ensures that the patch network goes to zero quickly when \( x \) is away from \( \mathcal{A} \). However, it still makes a small change on the neighbors of \( \mathcal{A} \). The following theorem shows that for a big enough \( \gamma \), the patch network would not change a correct region into incorrect.

Theorem 3 (Limited Side Effect). Given a correctness property \( \Phi = (\Phi_{in}, \Phi_{out}) \), a patch region \( \mathcal{A} \) and the corresponding patch network \( h(x, \gamma) \), there exists a positive number \( \Gamma \) such that for any \( \gamma \geq \Gamma \), we have

1. for any linear region \( \mathcal{B} \), if \( \mathcal{B} \cap \mathcal{A} = \emptyset \), then \( \hat{f}(x, \gamma) = f(x) \);
2. for any linear region \( \mathcal{C} \) who is a neighbor of \( \mathcal{A} \) \( (\mathcal{C} \cap \mathcal{A} \neq \emptyset) \), if \( f|_{\mathcal{C}} \models \Phi \), then \( \hat{f}|_{\mathcal{C}}(x, \gamma) \models \Phi \).

Corollary 1 (Incremental Repair). For multiple-region repair, the patch for a new region \( \mathcal{A}' \) would not cause a previous patched region \( \mathcal{A} \) to become incorrect.

4.3 Minimum Repair

Theorem 4 (Minimum Repair). For any ReLU DNN \( \hat{f} \), which is linear on a patch region \( \mathcal{A} \) and satisfies the specification \( \Phi \), there exists a positive number \( \Gamma \), such that for all \( \gamma \geq \Gamma \),

\[
\max_{x \in \mathcal{X}} |\hat{f}(x) - f(x)| \geq \max_{x \in \mathcal{X}} |h_{\mathcal{A}}(x, \gamma)|.
\]

4.4 Efficiency

Theorem 5 (Polynomial-Time Efficiency). REASSURE terminates in polynomial-time in the size of the neural network and the number of buggy linear regions.

Algorithm 1 REASSURE

Input: A specification \( \Phi = (\Phi_{in}, \Phi_{out}) \), a ReLU DNN \( f \) and a set of buggy points \( \{ \bar{x}_1, \ldots, \bar{x}_L \} \subset \Phi_{in} \).
Output: A repaired ReLU DNN \( \hat{f} \).

1: for \( l = 1 \) to \( L \) do
2: Generate the patch area \( \mathcal{A}_l \) from buggy point \( \bar{x}_l \) according to Equation (1);
3: Generate a support network \( g_{\mathcal{A}_l} \) according to Equation (2);
4: Solve the linear programming problem (6) to find the optimal affine patch network \( p_{\mathcal{A}_l} \);
5: end for
6: Combine all support networks \( g_{\mathcal{A}_l} \) and the corresponding patch networks \( p_{\mathcal{A}_l} \) to get the overall patch network \( h \) according to Equation (11).
7: return \( \hat{f} = f + h \)
5 Experiments

In this Section, we compare REASSURE with state-of-the-art methods on both point-wise repairs and area repairs. The experiments were designed to answer the following questions:

Q1 Effectiveness: How effective is a repair in removing known buggy behaviors?
Q2 Locality: How much side effect (i.e. modification outside the patch area in the function space) does a repair produce?
Q3 Function Change: How much does a repair change the original neural network in the function space?
Q4 Performance: Whether and how much does a repair adversely affect the overall performance of the neural network?

We consider the following evaluation criteria:

1. Efficacy (E): % of given buggy points or buggy linear regions that are repaired.
2. Norm Difference (ND): average norm ($L_\infty$ or $L_2$) difference between the original DNN and the repaired DNN on a set of inputs (e.g. training and testing data; more details in the tables). We use ND to measure how a repair change the original neural network on function space. Note that the maximum $L_\infty$ norm difference is 1 and the maximum $L_2$ norm difference is $\sqrt{2}$.
3. Norm Difference on Patch Area (NDP): average norm ($L_\infty$ or $L_2$) difference between the original DNN and the repaired DNN on patch areas (calculated on random sampled points on patch areas or near the buggy points; details in the tables). We use NDP to measure the locality of a repair.
4. Accuracy (Acc): accuracy on training or testing data to measure the extent to which a repair preserves the performance of the original neural network.
5. Negative Side Effect (NSE): NSE is only for area repair. It is the percentage of correct linear regions (outside of patch area) that become incorrect after a repair. If a repair has a nonzero NSE, the new repair may invalidate a previous repair and lead to a circular repair problem.

All experiments were run on an Intel Core i5 @ 3.4 GHz with 32 GB of memory. We use Gurobi Gurobi Optimization, LLC [2021] to solve the linear programs.

We compared REASSURE with the representative related works in Table 1. REASSURE, MDNN and PRDNN guarantee to repair all the buggy points (linear regions). Retrain and Fine-Tuning cannot guarantee 100% efficacy in general and we run them until all the buggy points are repaired.

5.1 Point-wise Repairs: MNIST

We train a ReLU DNN on the MNIST dataset LeCun [1998] as the target DNN. It is a multilayer perceptron with ReLU activation functions. It has an input layer with 784 nodes, 2 hidden layers with 256 nodes in each layer, and a final output layer with 10 nodes.

The goal of a repair is to fix the behaviors of the target DNN on buggy inputs that are found in the test dataset. Thus, the repaired DNN is expected to produce correct predictions for all the buggy inputs.

The results are shown in Table 2. REASSURE achieves almost zero modification outside the patch area (ND) amongst all four methods. In addition, REASSURE produces the smallest modification on the patch area (NDP) and preserves the performance of the original DNN (almost no drop on Acc).

We also compare REASSURE with MDNN on the watermark removal experiment from their paper. We were not able to run the code provided in the MDNN Github repository, but we were able to run the target DNN models, watermarked images, and MDNN-repaired models from the same repository.

The target DNN is from Goldberger et al. [2020], which has an input layer with 784 nodes, a single hidden layer with 150 nodes, and a final output layer with 10 nodes. The target DNN is watermarked by the method proposed in Adi et al. [2018] on a set of randomly chosen images $x_i$ with label $f(x_i)$.

The goal is to change the DNN’s predictions on all watermarks $x_i$ to any other label $y \neq f(x_i)$ while preserving the DNN’s performance on the MNIST test data. For REASSURE, we set the prediction $y = f(x_i) - 1$ if $f(x_i) > 1$, and $y = 10$ otherwise.
We use Specification 1, which is similar to Property 5 in Katz et al. [2017]. We use the method from Girard-Satabin et al. [2021] to compute all the linear regions for $N_{1,4}$ in the area $\Phi_{in}$ of Specification 1 and a total of 87 buggy linear regions were found. We apply both REASSURE and PRDNN to repair these buggy linear regions.

**Specification 1.** If the intruder is near and approaching from the left, the network advises "strong right."

*Input constraints:* $\Phi_{in} = \{(x, y, \psi) | 10 \leq x \leq 5000, 10 \leq y \leq 5000, -\pi \leq \psi \leq -1/2\pi\}$. *Output constraint:* $f(x, y, \psi)_i \geq f(x, y, \psi)_{i-1}$ for $i = 0, 1, 2, 3$.

---

**Table 2:** Point-wise Repairs on MNIST. We use the first hidden layer as the repair layer for PRDNN. The test accuracy

| #P | ND($L_\infty$) | ND($L_2$) | Fine-Tuning ND($L_\infty$) | ND($L_2$) | Acc | ND($L_\infty$) | ND($L_2$) | PRDNN ND($L_\infty$) | ND($L_2$) | Acc |
|----|----------------|-----------|---------------------------|-----------|-----|----------------|-----------|----------------------|-----------|-----|
| 10 | 0.0000         | 0.0000    | 0.0900                    | 0.128     | 96.8%| 0.089          | 0.126     | 0.071                | 0.087     | 87.5%|
| 5  | 0.0000         | 0.0000    | 0.127                     | 0.180     | 96.8%| 0.481          | 0.681     | 0.443                | 0.562     | 57.1%|
| 25 | 0.0000         | 0.0000    | 0.299                     | 0.420     | 96.8%| 0.904          | 1.278     | 0.637                | 0.815     | 6.7% |
| 50 | 0.0000         | 0.0000    | 0.429                     | 0.602     | 96.8%| 0.925          | 1.308     | 0.821                | 1.024     | 4.8% |
| 100 | 0.0000       | 0.0000    | 0.462                     | 0.648     | 96.8%| 0.955          | 1.350     | 0.909                | 1.083     | 5.1% |

The results are shown in Table 3. Both REASSURE and MDNN remove all the watermarks. However, MDNN introduces significant distortion to the target DNN and as a result the test accuracy drops rapidly as the number of repair points increases. In comparison, REASSURE removes all the watermarks with no harm to test accuracy.

**Table 3:** Watermark Removal. The test accuracy of the original DNN is 98.0%. #P: number of buggy points to repair. ND($L_\infty$), ND($L_2$): average ($L_\infty$, $L_2$) norm difference on both training and test data. NDP($L_\infty$), NDP($L_2$): average ($L_\infty$, $L_2$) norm difference on random sampled points near the buggy points. Acc: accuracy on test data. Note that REASSURE automatically performs area repairs on 784-dimensional inputs.

| #P | ND($L_\infty$) | ND($L_2$) | ND($L_\infty$) | ND($L_2$) | Acc   | ND($L_\infty$) | ND($L_2$) | ND($L_\infty$) | ND($L_2$) | Acc   |
|----|----------------|-----------|----------------|-----------|-------|----------------|-----------|----------------|-----------|-------|
| 10 | 0.0001         | 0.0002    | 0.2575         | 0.3495    | 98.1% | 0.0113        | 0.0155    | 0.7786         | 1.0801    | 98.1%|
| 20 | 0.0003         | 0.0003    | 0.1917         | 0.2637    | 98.2% | 0.0092        | 0.0126    | 0.7114         | 1.0757    | 98.4%|
| 50 | 0.0006         | 0.0006    | 0.2464         | 0.3555    | 98.5% | 0.0084        | 0.0115    | 0.8417         | 1.1584    | 98.7%|
| 100| 0.0011         | 0.0017    | 0.2540         | 0.3446    | 99.0% | 0.0084        | 0.0116    | 0.8483         | 1.1710    | 99.0%|

To the best of our knowledge, Sotoudeh and Thakur [2021] is the only other method that supports area repairs. In this experiment, we compare REASSURE with Sotoudeh and Thakur [2021] on an experiment where the setting is similar to the 2D Polytope ACAS Xu repair in their paper.

Sotoudeh and Thakur [2021] does not include a vertex enumeration tool (which is required for setting up their LP problem) in their code. We use pycddlib Troffaes [2018] to perform the vertex enumeration step when evaluating PRDNN. Note that the vertex enumeration tool does not affect the experimental results except running time.

We consider an area repair where the target DNN is the HCAS network (simplified version of ACAS Xu) $N_{1,4}$ (previous advisory equal to 1 and time to loss of vertical separation equal to 20s) from Julian and Kochenderfer [2019]. $N_{1,4}$ has an input layer with 3 nodes, 5 hidden layers with 25 nodes in each hidden layer, and a final output layer with 5 nodes. DNN outputs one of five possible control advisories ("Strong left", "Weak left", "Clear-of-Conflict", "Weak right" and "Strong right").

We use Specification 1, which is similar to Property 5 in Katz et al. [2017]. We use the method from Girard-Satabin et al. [2021] to compute all the linear regions for $N_{1,4}$ in the area $\Phi_{in}$ of Specification 1 and a total of 87 buggy linear regions were found. We apply both REASSURE and PRDNN to repair these buggy linear regions.

5.2 Area Repairs: HCAS

---

The technique in PRDNN for computing linear regions does not scale beyond two dimensions as stated in their paper. The input space of HCAS is 3D and that of ACAS Xu is 5D so we use HCAS in order to run their tool in our evaluation of area repairs.
We use AlexNet Krizhevsky et al. [2012] on the ImageNet dataset Russakovsky et al. [2015] as the target DNN. The resulting network has 650k neurons, consisting of five convolutional layers, some of which are followed by max-pooling layers, followed by five fully-connected layers with (9216, 4096, 512, 256, 256) nodes respectively. The total number of parameters in the resulting network is around 60 million.

The results are shown in Table 5. Both REASSURE and PRDNN successfully repair all the buggy points. REASSURE automatically performs area repair on point-wise repair problems. This means our area repair method scales well to high-dimensional polytopes (the input dimension of MNIST is 784) whereas PRDNN does not scale beyond 2D linear regions/polytopes.

5.3 Feature-Space Repairs: ImageNet

We use AlexNet Krizhevsky et al. [2012] on the ImageNet dataset Russakovsky et al. [2015] as the target DNN. The size of an input image is (224, 224, 3) and the total number of classes for ImageNet is 1000. We slightly modified AlexNet to simplify the evaluation: we only consider 10 output classes that our buggy images may lie on and use a multilayer perceptron with three hidden layers (512, 256, 256) respectively to mimic the last two layers of AlexNet. The resulting network has 650k neurons, consisting of five convolutional layers, some of which are followed by max-pooling layers, followed by five fully-connected layers with (9216, 4096, 512, 256, 256) nodes respectively. The total number of parameters in the resulting network is around 60 million.

The goal of the repair is to fix the behaviors of the target DNN on buggy inputs, which are found on ImageNet-A Hendrycks et al. [2021]. For REASSURE, we construct the patch network starting from the 17-th (third from the last) hidden layer (i.e. repair in a feature space).

The results are shown in Table 5. Both REASSURE and PRDNN successfully repair all the buggy points. REASSURE achieves almost zero modification on the validation images compared to the original DNN. In addition, REASSURE preserves the performance of the original DNN.
6 Discussion

6.1 Parameter Overhead

REASSURE introduces an additional network, patch network, and as a result adds new parameters to the original network. The average number of new parameters that REASSURE introduces per repair region is in $O(m|I|)$, where $m$ is the dimension of the target DNN’s input space and $|I|$ is the number of linear constraints for the H-representation of $A$. We can remove redundant constraints in this representation in polynomial time to make the additional network smaller, e.g. iteratively using an LP to check if a constraint is redundant. For the area repair experiment on HCAS, the average number of constraints for one linear region is 3.28 (which is only 3% of the original 125 constraints after removing the redundant constraints) and the average number of new parameters that REASSURE introduces per repair region is 66. As a comparison, the number of parameters in the original network is around 3000 and PRDNN doubles the number of parameters (as a result of the Decoupled DNN construction) regardless of the number of point-wise or area repairs. We further note that the removal of redundant constraints can be done offline as a post-repair optimization step.

Another way to cope with the additional space overhead is to leverage feature-space repairs. As described in Section 3.7, feature-space repairs allow us to construct the patch network starting from an intermediate layer. In addition, for most DNN structures, the dimension of an intermediate layer is typically smaller than the dimension of the input space. As a result, the number of additional parameters per repair region will be smaller. For the point-wise repair experiment on ImageNet, our feature-space repair adds 500k new parameters on average, which is only 0.1% of the additional parameters introduced if we were to construct the patch network starting from the input layer. With feature-space repairs, we are trading-off repair specificity, i.e. how localized the repair is in the input space, with parameter overhead. However, when the number of points or linear regions to repair becomes large, it may make sense to perform repairs in the feature space anyway for better generalization. We leave the in-depth investigation of feature-space repairs to future work.

6.2 Applying REASSURE To General CPWL Networks

Recall the result that an $\mathbb{R}^m \rightarrow \mathbb{R}$ function is representable by a ReLU DNN if and only if it is a continuous piecewise linear (CPWL) function Arora et al. [2016]. We use convolutional neural networks as an example to show how REASSURE can be applied to more general CPWL networks. Convolutional neural networks (CNNs) are neural networks with convolution layers and maxpooling layers. For simplicity, we assume the CNNs also use ReLU activation functions (but in general other CPWL activation functions will also work). The convolutional layers can be viewed as special linear layers. The maxpooling layers can be converted to linear operations with ReLU activation functions as follows.

$$
\begin{align*}
\max(x_1, x_2, ..., x_n) &= \max(x_1, \max(x_2, x_3, ..., x_n)) \\
\max(x_i, x_j) &= \max(x_i - x_j, 0) + x_j = \sigma(x_i - x_j) + x_j
\end{align*}
$$

where $\sigma$ is the ReLU activation function. Thus, REASSURE can be used to repair CNNs as well.

7 Conclusion

We have presented a novel approach for repairing ReLU DNNs with strong theoretical guarantees. Across a set of benchmarks, our approach significantly outperforms existing methods in terms of efficacy, locality, and limiting negative side effects. Future directions include further investigation on feature-space repairs and identifying a lower-bound for $\gamma$.

References

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. CoRR, abs/1604.07316, 2016. 1

Nida Shahid, Tim Rappon, and Whitney Berta. Applications of artificial neural networks in health care organizational decision-making: A scoping review. PLOS ONE, 14(2):1–22, 02 2019. doi:10.1371/journal.pone.0212356. URL https://doi.org/10.1371/journal.pone.0212356. 1

Kyle D Julian, Mykel J Kochenderfer, and Michael P Owen. Deep neural network compression for aircraft collision avoidance systems. Journal of Guidance, Control, and Dynamics, 42(3):598–608, 2019. 1

Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Kurt Keutzer, Alberto Sangiovanni-Vincentelli, and Sanjit A Seshia. Counterexample-guided data augmentation. arXiv preprint arXiv:1805.06962, 2018. 1
Xuhong Ren, Bing Yu, Hua Qi, Felix Juefei-Xu, Zhuo Li, Wanli Xue, Lei Ma, and Jianjun Zhao. Few-shot guided mix for dnn repairing. In 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 717–721. IEEE, 2020.

Anton Sinitsin, Vsevolod Plokhotnyuk, Dmitriy Pyrkin, Sergei Popov, and Artem Babenko. Editable neural networks. arXiv preprint arXiv:2004.00345, 2020.

Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. Mode: automated neural network model debugging via state differential analysis and input selection. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 175–186, 2018.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. Proceedings of the National Academy of Sciences, 114(13):3521–3526, 2017. ISSN 0027-8424. doi:10.1073/pnas.1611835114. URL https://www.pnas.org/content/114/13/3521.

Guoliang Dong, Jun Sun, Jingyi Wang, Xinyu Wang, and Ting Dai. Towards repairing neural networks correctly. arXiv preprint arXiv:2012.01872, 2020.

Ben Goldberger, Guy Katz, Yossi Adi, and Joseph Keshet. Minimal modifications of deep neural networks using verification. In LPAR, volume 2020, page 23rd, 2020.

Matthew Sotoudeh and Aditya V Thakur. Provable repair of deep neural networks. In Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, pages 588–603, 2021.

Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. CoRR, abs/1611.01491, 2016. URL http://arxiv.org/abs/1611.01491.

Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. CoRR, abs/1711.02114, 2017. URL http://arxiv.org/abs/1711.02114.

Guang-He Lee, David Alvarez-Melis, and Tommi S Jaakkola. Towards robust, locally linear deep networks. arXiv preprint arXiv:1907.03207, 2019.

Martin Henk, Jürgen Richter-Gebert, and Günter M. Ziegler. Basic properties of convex polytopes. In HANDBOOK OF DISCRETE AND COMPUTATIONAL GEOMETRY, CHAPTER 13, pages 243–270. CRC Press, Boca, 1997.

Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. Robust optimization, volume 28. Princeton university press, 2009.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021. URL https://www.gurobi.com.

Yann LeCun. The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist/, 1998.

Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In 27th {USENIX} Security Symposium ({USENIX} Security 18), pages 1615–1631, 2018.

Matthias Troffaes. pycddlib—a python wrapper for komei fukudals cddlib, 2018.

Kyle D Julian and Mykel J Kochenderfer. Guaranteeing safety for neural network-based aircraft collision avoidance systems. In 2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC), pages 1–10. IEEE, 2019.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In International Conference on Computer Aided Verification, pages 97–117. Springer, 2017.

Julien Girard-Satabin, Aymeric Varasse, Marc Schoenauer, Guillaume Charpiat, and Zakaria Chihani. Disco verification: Division of input space into convex polytopes for neural network verification. arXiv preprint arXiv:2105.07776, 2021.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25:1097–1105, 2012.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV), 115(3):211–252, 2015. doi:10.1007/s11263-015-0816-y.
8 Appendix

8.1 An Alternative LP Solution

In Section 3.4, we show that optimization problem (3) can be converted to an LP via robust optimization Ben-Tal et al. [2009]. However, taking the dual of the inner LP in robust optimization introduces new variables. Specifically, the number of new variables is in \( \mathcal{O}(n|I|) \), where \( n \) is the DNN’s output dimension and \( |I| \) is the number of constraints for the linear region. Thus, it is more expensive to solve the LP when \( n \) is large (although still much less expensive than enumerating all the vertices). In this section, we show that we can solve optimization problem (3) via LP more efficiently for many useful cases such as the classification problem in Example 1 and the HCAS example in Section 5.2.

We consider the case where \( \Phi_{out} \) can be expressed as \( \{ y \mid q_i \leq Py \leq q_u \} \) where \( P \) is a full row rank matrix and \(-\infty \leq q_i[i] \leq q_u[i] \leq +\infty \) (\( q_i[i] \) and \( q_u[i] \) are the \( i \)-th elements of \( q_i \) and \( q_u \) respectively).

Consider the following optimization problem.

\[
\begin{align*}
\min_T & \max_{x \in \mathcal{A}} |T(f(x)) - f(x)| \\
q_i & \leq P(T(f(x))) \leq q_u, \forall x \in \mathcal{A}
\end{align*}
\]  

(13)

where \( T : \mathbb{R}^n \rightarrow \mathbb{R}^n \) is a linear transformation on the DNN’s output space \( \mathbb{R}^n \).

**Theorem 6.** On linear region \( \mathcal{A} \), we have \( f|\mathcal{A}(x) = f_1x + f_2 \) for some matrix \( f_1 \) and vector \( f_2 \). Assuming that \( f_1 \) is full rank\(^3\), the optimization problem in (3) and the optimization problem in (13) are equivalent.

Note that \( q_i \leq P(T(f(x))) \leq q_u \) can be achieved row by row. Thus, we can find a one-dimensional linear transformation via LPs and combine them into a single linear transformation to solve optimization problem (13).

For every row of \( P \), say the \( i \)-th row, we can check the lower bound and upper bound of \( \{P(f(x)) \mid \forall x \in \mathcal{A}\} \) on the \( i \)-th dimension by solving the following LP problems

\[
\begin{align*}
lb[i] &= \min_{x \in \mathcal{A}} P[i](f(x)) \\
ub[i] &= \max_{x \in \mathcal{A}} P[i](f(x))
\end{align*}
\]  

(14)

where \( P[i] \) is the \( i \)-th row of \( P \).

Then for each row \( i \), we take a minimal linear transformation \( V[i](x) = v_1[i](x) + v_2[i] \) to transfer interval \( [lb[i], ub[i]] \) inside interval \( [q_l[i], q_u[i]] \). We can take \( v_1[i] = 1 \) if \( q_u[i] - q_l[i] \) is large (although still much less expensive than enumerating all the vertices). In this case, \( v_1[i] = \frac{q_u[i] - q_l[i]}{ub[i] - lb[i]} \). And \( v_2[i] = q_l[i] - v_1[i]lb[i] \) if \( |q_l[i] - v_1[i]lb[i]| \leq |lb[i]ub[i] - q_u[i]| \), else \( v_1[i]ub[i] - q_u[i] \).

Since matrix \( P \) is full row rank, we can find a linear transformation \( T \) that is equivalent to \( V \):

\[
T = \hat{P}^{-1}V\hat{P} \Rightarrow P(T(f(x))) = V(P(f(x)))
\]  

(15)

where \( \hat{P} = \left[ \begin{array}{cc} P & P_\perp \end{array} \right] \) is an orthogonal extension of \( P \) (\( P \) and \( P_\perp \) are orthogonal to each other and \( \hat{P} \) is a full rank square matrix).

Once we have \( T \), we can obtain an affine patch function \( p_\mathcal{A}(x) = T(f(x)) - f(x) \).

8.2 Proofs of Theorems

We prove Theorem 1 after Corollary 1, since the proof of Theorem 1 uses the result of Corollary 1.

**Lemma 2.** The repaired DNN \( \hat{f} \) returned by REASSURE is guaranteed to satisfy the specification \( \Phi \) on patch area \( \mathcal{A} \) in single-region repair case.

**Proof.** By the definition of \( p_\mathcal{A} \), we have \( f(x) + p_\mathcal{A}(x) \in \Phi_{out} \) for all \( x \in \mathcal{A} \).

\(^3\)Note that for neural networks that are trained by a stochastic method, with probability one \( f_1 \) is full rank.
For any \( x \in A \), we have \( g_A(x, \gamma) = 1 \) and \( h_A(x, \gamma) = \sigma(p_A(x)) - \sigma(-p_A(x)) = p_A(x) \). Therefore,\[
\hat{f}(x) = f(x) + h_A(x, \gamma) = f(x) + p_A(x) \in \Phi_{\text{out}} \tag{16}
\]
Thus, the patched neural network \( \hat{f} \) meets the specification \( \Phi \) on \( A \).

**Theorem 2** (Completeness). **REEASURE** can always find a solution to the minimal point-wise repair or the minimal area repair problem.

**Proof.** For every patch area \( A \), we can always find a support network \( g_A \). For any \( \Phi_{\text{out}} \) and \( A \), there exists an affine function \( p_A \) such that \( p_A(x) \in \Phi_{\text{out}}, \forall x \in A \). Therefore, the LP (6) is always feasible and **REEASURE** can find an affine patch function \( p_A \).

Once we have \( g_A \) and \( p_A \) for patch area \( A \), **REEASURE** returns a patch network either by Equation (10) or by Equation (11).

**Theorem 3** (Limited Side Effect). Given a correctness property \( \Phi = (\Phi_{\text{in}}, \Phi_{\text{out}}) \), a patch region \( A \) and the corresponding patch network \( h(x, \gamma) \), there exists a positive number \( \Gamma \) such that for any \( \gamma \geq \Gamma \), we have

1. for any linear region \( B \), if \( B \cap A = \emptyset \), then \( \hat{f}(x, \gamma) = f(x) \);
2. for any linear region \( C \) who is a neighbor of \( A (C \cap A \neq \emptyset) \), if \( f|_C = \Phi \), then \( \hat{f}(x, \gamma) = \Phi \).

**Proof.** Since a multi-region repair is a composition of multiple single-region repairs according to Equation (11), we can prove the limited side effect of a multi-region repair by proving the limited side effect of its constituent single-region repairs. Below, we prove the limited side effect of a single-region repair.

Consider patch area \( A = \{x | a_i x \leq b_i, i \in I\} \) and \( A_{>0}(\gamma) = \{x | h(x, \gamma) > 0\} \).

1. Since the number of neighbors for \( A \) are finite, we can take a big enough \( \gamma \), such that for any \( B \), if \( B \cap A = \emptyset \), \( B \cap A_{>0}(\gamma) = \emptyset \). Thus, we have \( \hat{f}(x, \gamma) = f(x) \) on \( B \).

2. For any linear region \( C \) who is a neighbor of \( A \), i.e. \( C \neq A \) and \( C \cap A \neq \emptyset \). \( \hat{f} \) is no longer a linear function on \( C \), since there are some hyperplanes introduced by our repair that will divide \( C \) into multiple linear regions.

Specifically, those hyperplanes are \( \{x | \gamma (a_i x - b_i) + 1 = 0\} \) for \( i \in I \), \( \{x | \sum_{i \in I} g(a_i x - b_i, \gamma) - |I| + 1 = 0\} \), \( \{x | p(x) + K \cdot g_A(x, \gamma) - K = 0\} \) and \( \{x | -p(x) + K \cdot g_A(x, \gamma) - K = 0\} \).

For any point \( x \) in those hyperplanes, it will fall into one of the following four cases.

(a) \( x \in \{x | \gamma (a_i x - b_i) + 1 = 0\} \) for some \( i \in I \), then \( g_A(x, \gamma) = 0 \), \( h(x, \gamma) = 0 \) and \( \hat{f}(x) \in \Phi_{\text{out}} \);
(b) \( x \in \{x | \sum_{i \in I} g(a_i x - b_i, \gamma) - |I| + 1 = 0\} \), then \( g_A(x, \gamma) = 0 \), \( h(x, \gamma) = 0 \) and \( \hat{f}(x) \in \Phi_{\text{out}} \);
(c) \( x \in \{x | p(x) + K \cdot g_A(x, \gamma) - K = 0\} \), then \( p(x) = K - K \cdot g_A(x, \gamma) \geq 0, -p(x) + K \cdot g_A(x, \gamma) - K \leq 0, h(x, \gamma) = 0 \) and \( \hat{f}(x) \in \Phi_{\text{out}} \);
(d) \( x \in \{x | -p(x) + K \cdot g_A(x, \gamma) - K \} \), then \( p(x) = K \cdot g_A(x, \gamma) - K \leq 0, p(x) + K \cdot g_A(x, \gamma) - K \leq 0, h(x, \gamma) = 0 \) and \( \hat{f}(x) \in \Phi_{\text{out}} \).

By the above analysis, we have \( \hat{f}(x) \in \Phi_{\text{out}} \) for the boundary of the new linear regions. Since \( \hat{f} \) is linear on the new linear regions and \( \Phi_{\text{out}} \) is convex, \( \hat{f}(x) \in \Phi_{\text{out}} \) for any \( x \in C \).

**Remark:** By Theorem 3, we have that a patch would not change a correct linear region to an incorrect one.

**Corollary 1** (Incremental Repair). For multiple-region repair, the patch for a new region \( A' \) would not cause a previous patched region \( A \) to become incorrect.

**Proof.** After applying the patch to linear region \( A \), we have that the resulting network is correct on \( A \). When applying a new patch to another linear region \( A' \), by Theorem 3, the new patch would not make a correct linear region \( A \) incorrect.
**Theorem 1** (Soundness). The repaired DNN $\hat{f}$ returned by REASSURE is guaranteed to satisfy the specification $\Phi$.

**Proof.** The proof has two parts:

1. to show that $\hat{f}$ satisfies the specification $\Phi$ on $A$, and
2. to show that $\hat{f}$ satisfies the specification $\Phi$ outside of $A$.

Part 1:
Lemma (2) shows $\hat{f}$ satisfies the specification $\Phi$ for single-region repair on $A$.

For the multi-region case, consider a set of buggy linear regions $\bigcup_{1 \leq i \leq I} A_i$ with the corresponding support neural network $g_{A_i}$ and affine patch function $p_{A_i}$ for each $A_i$. For the multi-region repair construction in Equation (11), we refer to $\sigma(p_{A_j} - p_{A_{j-1}} + \max_{k \geq j} \{g_{A_k}\} K_j - K_j)$ as the $j$-th patch and $f_j = f + \sum_{j' \leq j} \sigma(p_{A_{j'}} - p_{A_{j'-1}} + \max_{k \geq j'} \{g_{A_k}\} K_j - K_j)$ as the network after the $j$-th patch.

For any $x$ in patch area $\bigcup_{1 \leq i \leq I} A_i$, we can find a $j$ such that $x \in A_j$ but $x \notin A_k$ for all $k > j$. After the first $j$ patches $\sigma(p_{A_1}(x) + \max_{k \geq 1} \{g_{A_k}(x, \gamma)\} K_1 - K_1), \sigma(p_{A_2}(x) - p_{A_1}(x) + \max_{k \geq 2} \{g_{A_k}(x, \gamma)\} K_2 - K_2), \ldots, \sigma(p_{A_j}(x) - p_{A_{j-1}}(x) + \max_{k \geq j} \{g_{A_k}(x, \gamma)\} K_j - K_j)$, the DNN’s output at $x$ becomes $\hat{f}_j(x) = f(x) + p_{A_j}(x)$ which meets our specification $\Phi$ at $x$ by the definition of $p_{A_j}(x)$.

Since $x \notin A_k$ for all $k > j$, then by Corollary 1, the rest of the patches would not change a correct area to an incorrect area. Therefore, we have the final patched neural network $\hat{f}$ meets specification $\Phi$ on $\bigcup_{1 \leq i \leq I} A_i$.

Part 2:
To show that $\hat{f}$ satisfies $\Phi$ outside of $A$.

For any $x$ outside the patch area $\bigcup_{1 \leq i \leq I} A_i$, we have $x$ lies on a correct linear region (linear region that satisfies the specification $\Phi$). By Theorem 3, we have either $\hat{f}(x) = f(x)$ or $\hat{f}(x) \in \Phi_{out}$. Therefore, $\hat{f}$ satisfies $\Phi$ outside of $A$.

**Theorem 4** (Minimum Repair). For any ReLU DNN $\hat{f}$, which is linear on a patch region $A$ and satisfies the specification $\Phi$, there exists a positive number $\Gamma$, such that for all $\gamma \geq \Gamma$,

$$\max_{x \in X} |\hat{f}(x) - f(x)| \geq \max_{x \in X} |h_{A}(x, \gamma)|. \quad (17)$$

**Proof.** We consider the general case where the linear patch function is obtained from Equation (3).

For any DNN $\hat{f}$, which is linear on patch region $A$ and satisfies the specification $\Phi$, we have $\max_{x \in A} |\hat{f} - f| \geq \max_{x \in A} |cx + d| = \max_{x \in A} |h_{A}(\cdot, \gamma)|$ on patch area $A$ by Equation (3).

Therefore, we only need to show:

$$\max_{x \notin A} |h_{A}(\cdot, \gamma)| \leq \max_{x \in A} |h_{A}(\cdot, \gamma)| \quad (18)$$

Since parameter $\gamma$ controls the slope of $h_{A}(\cdot, \gamma)$ outside of patch area $A$, a large $\gamma$ means that $h_{A}(\cdot, \gamma)$ will drop to zero quickly outside of $A$. Therefore, we can choose a large enough $\Gamma$ such that $h_{A}(\cdot, \gamma)$ drops to zero faster than the change of linear patch function $cx + d$.

Therefore, we have that for any $\gamma \geq \Gamma$,

$$\max_{x \notin A} |h_{A}(\cdot, \gamma)| \leq \max_{x \in A} |h_{A}(\cdot, \gamma)| = \max_{x \in X} |h_{A}(\cdot, \gamma)| \leq \max_{x \in A} |\hat{f} - f| \leq \max_{x \in X} |\hat{f} - f| \quad (19)$$

**Theorem 5** (Polynomial-Time Efficiency). REASSURE terminates in polynomial-time in the size of the neural network and the number of buggy linear regions.
Proof. We consider the affine patch function solved via Robust Optimization (9). Suppose \( \mathcal{A} = \{ x \in X \mid a_i x \leq b_i, i \in I \} \). The running time for solving a Linear Programming is polynomial in the number of variables, and the number of variables for Linear Programming (9) is polynomial in \( |I| \), which is the number of constraints for \( \mathcal{A} \), the DNN’s input dimension, and the DNN’s output dimension. Since \( |I| \) is polynomial in the size of the DNN, REASSURE runs in polynomial time in the size of the neural network.

In addition, since REASSURE computes the support network \( g_{\mathcal{A}} \) and affine patch function \( p_{\mathcal{A}} \) for each \( \mathcal{A} \) one by one (see Algorithm 1), the time complexity of REASSURE is linear in the number of buggy linear regions.

\[ \Box \]

**Theorem 6.** On linear region \( \mathcal{A} \), we have \( f|_{\mathcal{A}}(x) = f_1 x + f_2 \) for some matrix \( f_1 \) and vector \( f_2 \). Assuming that \( f_1 \) is full rank\(^4\), the optimization problem in (3) and the optimization problem in (13) are equivalent.

Proof. On one side, for any \( c, d \), since \( f_1 \) is full rank, there exists a linear transformation \( T \), such that \( T(f(x)) = T(f_1 x + f_2) = (f_1 + c)x + (f_2 + d) = f(x) + cx + d \).

On the other side, for any \( T \), since \( T(f(x)) - f(x) \) is linear, there exist \( c, d \), such that \( cx + d = T(f(x)) - f(x) \). \[ \Box \]

### 8.3 Additional Experiment Details

#### Area Repair: HCAS

Table 6 is the comparison with PRDNN using the last layer as the repair layer. All other settings are the same as those in Section 5.2.

| #A | ND\((L_{\infty})\) | NDP\((L_{\infty})\) | NSE | Acc | T       | ND\((L_{\infty})\) | NDP\((L_{\infty})\) | NSE | Acc | T       |
|----|-----------------|-----------------|-----|-----|--------|-----------------|-----------------|-----|-----|--------|
| 10 | 2.662e-05       | 1.318e-05       | 0%  | 98.1% | 1.0422 | 0.0030          | 0.205           | 16% | 71.4% | 2.90+0.100 |
| 20 | 2.918e-05       | 0.022           | 0%  | 98.1% | 1.1856 | 0.0031          | 0.467           | 66% | 70.5% | 5.81+0.169 |
| 50 | 8.289e-05       | 0.176           | 0%  | 98.1% | 1.8174 | 0.0031          | 0.467           | 66% | 70.5% | 14.54+0.353 |
| 87 | 0.0004          | 0.459           | 0%  | 97.8% | 2.4571 | 0.0031          | 0.467           | 66% | 70.5% | 25.30+0.467 |

Table 6: Area Repairs on HCAS. We use the the last hidden layer as the repair layer for PRDNN. The test accuracy of the original DNN is 97.9%. #A: number of buggy linear regions to repair; ND\((L_{\infty})\): average \( L_{\infty} \) norm difference on training data; NDP\((L_{\infty})\): average \( L_{\infty} \) norm difference on random sampled data on input constraints of Specification 1; NSE: % of correct linear regions that is repaired to incorrect; Acc: accuracy on training data (no testing data available); T: running time in seconds. For PRDNN, the first running time is for enumerating all the vertices of the polytopes and the second is for solving the LP problem in PRDNN.

#### Hyperparameters used in Repair:

We set learning rate to \( 10^{-3} \) for Retrain in the point-wise repair experiment.

We set learning rate to \( 10^{-2} \) and momentum to 0.9 for Fine-Tuning in the point-wise repair experiment.

PRDNN requires specifying a layer for weight modification. We use the first hidden layer as the repair layer, which has the best performance in our experiment settings, unless otherwise specified.

\[ ^4 \text{Note that for neural networks that are trained by a stochastic method, with probability one } f_1 \text{ is full rank.} \]