Neural Network Repair with Reachability Analysis

Xiaodong Yang[1], Tom Yamaguchi[2], Hoang-Dung Tran[3], Bardh Hoxha[2], Taylor T Johnson[1], and Danil Prokhorov[2]

[1] Vanderbilt University, Nashville, TN, USA
[2] TRINA, Toyota NA R&D, Ann Arbor, MI, USA
[3] University of Nebraska, Lincoln, NE, USA

Abstract. Safety is a critical concern for the next generation of autonomy that is likely to rely heavily on deep neural networks for perception and control. This paper proposes a method to repair unsafe ReLU DNNs in safety-critical systems using reachability analysis. Our repair method uses reachability analysis to calculate the unsafe reachable domain of a DNN, and then uses a novel loss function to construct its distance to the safe domain during the retraining process. Since subtle changes of the DNN parameters can cause unexpected performance degradation, we also present a minimal repair approach where the DNN deviation is minimized. Furthermore, we explore applications of our method to repair DNN agents in deep reinforcement learning (DRL) with seamless integration with learning algorithms. Our method is evaluated on the ACAS Xu benchmark and a rocket lander system against the state-of-the-art method ART. Experimental results show that our repair approach can generate provably safe DNNs on multiple safety specifications with negligible performance degradation, even in the absence of training data.[4].

Keywords: Neural network repair · reachability analysis.

1 Introduction

Although deep neural networks (DNNs) have been successful in many areas, their trustworthiness remains a primary issue preventing widespread use. Recently, many techniques for analyzing behaviors of DNNs have been presented [9, 6, 14, 19]. Given a DNN, these works present post-training verification methods that generate a safety certificate over input-output specifications. One challenge that remains is the repair problem, where given a DNN with erroneous behaviors, an automatic process repairs the network with respect to the specification.

Existing works that improve the safety and robustness of DNNs can be classified into two main categories. The first category relies on singular adversarial inputs to make specialized modifications on neural weights that likely cause misbehavior. In [13], the paper presents a technique named *Arachne*. There, given a

[4] Code is available online at https://github.com/Shaddadi/veritex.git

set of finite adversarial inputs, with the guidance of a fitness function, *Arachne* searches and subsequently modifies neural weights that are likely related to these undesired behaviors. In [2], the paper proposes a DNN verification-based method that modifies undesirable behavior of DNNs by manipulating neural weights of the output layer. The correctness of the repaired DNN is then proved with a verification technique. In [15], the repair approach first localizes the potential faulty DNN parameter at an intermediate layer or the last layer, and then conducts a small modification using constraint solving. In [11], the method poses the repair problem as a mixed-integer quadratic program to adjust the parameter of a single layer, such that undesired behaviors can be reduced and meanwhile the change in DNNs is minimized. However, these methods only enhance the robustness of DNNs, meaning that provably safe DNNs cannot be generated. In addition, the modification of weights based on individual adversarial examples may not capture the impact on the whole performance of the network.

The second category is based on adversarial training, such as [3, 10]. However, these methods do not provide guarantees regarding the safety of the DNN. To solve this issue, some works incorporate reachability analysis in this process, such that they can train a model that is provably safe on a norm-bounded domain [17, 12, 8]. Given a norm-bounded input range, these approaches over approximate the output reachable domain of DNNs with a convex region. Then they minimize the worst-case loss over these regions, which aims to migrate all unsafe outputs to the desired domain. The primary issue of these approaches is that the approximation error accumulates during computation. For large input domains or complex DNNs, their approximated domain can be so conservative that a low-fidelity worst-case loss may result in significant accuracy degradation. We confirm this issue through experiments in comparison with ART [8].

In this paper, we propose a repair method for ReLU DNNs based on exact reachability analysis. Compared to over-approximation approaches, the exact analysis enables us to precisely compute the unsafe reachable domain of a DNN and its distance to the safe domain. In the repair process, this distance is constructed with a loss function for minimization. Additionally, by combining it with another objective function that minimizes the change of the DNN parameters, a minimal-repaired DNN can be learned, which aims to preserve the performance. Experiments indicate that our method can successfully repair unsafe DNNs on multiple safety specifications with negligible impact on performance.

2 Deep Neural Network Repair

This section provides definitions and problem statements for DNN repair, and a brief overview of reachability analysis for DNNs. The problem is also extended to repair DNN agents in deep reinforcement learning.

2.1 Provably Safe DNNs

Let $f_\theta : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{X} \subseteq \mathbb{R}^{|\mathbf{x}|}$ and $\mathcal{Y} \subseteq \mathbb{R}^{|\mathbf{y}|}$ denote the input and output space of a DNN with the parameter set θ, where given an input $\mathbf{x} \in \mathcal{X}$, it produces an output $\mathbf{y} = f_\theta(\mathbf{x}) \in \mathcal{Y}$. The safety problem of DNNs with reachability analysis on safety properties is formally defined as follows.

Definition 1 (Safety Property). *A safety property $\mathcal{P}$ of a DNN f_θ specifies a bounded input domain $\mathcal{I} \subseteq \mathcal{X}$ and a corresponding undesired output domain $\mathcal{U} \subseteq \mathcal{Y}$. The domain $\mathcal{I}$ refers to an interval with the lower bound $\underline{\boldsymbol{x}}$ and the upper bound $\overline{\boldsymbol{x}}$. The domain $\mathcal{U}$ refers to either a convex domain $A\boldsymbol{y} + b \leq 0$ or a non-convex domain consisting of multiple such convex domains.*

Definition 2 (DNN Reachable Domain). *Given an input domain $\mathcal{I} \subseteq \mathcal{X}$ to a DNN f_θ, its output reachable domain will be a subspace $\mathcal{O} \subseteq \mathcal{Y}$ where $\forall \boldsymbol{x} \in \mathcal{I}$, its output $\boldsymbol{y} = f_\theta(\boldsymbol{x})$ and $\boldsymbol{y} \in \mathcal{O}$. The domain $\mathcal{O}$ is exact if it only contains the output of $\boldsymbol{x} \in \mathcal{I}$. Otherwise, it is over approximated. It is formulated as $\mathcal{O} = \mathbb{N}(\mathcal{I})$.*

Definition 3 (DNN Safety Verification). *A DNN f_θ is safe on a safety property $\mathcal{P}$ that specifies an input domain $\mathcal{I}$ and an output unsafe domain $\mathcal{U}$, or $f_\theta \models \mathcal{P}$, if the exact output reachable domain $\mathcal{O} = \mathbb{N}(\mathcal{I})$ satisfies that $\mathcal{O} \cap \mathcal{U} = \emptyset$. Otherwise, it is unsafe, or $f_\theta \not\models \mathcal{P}$.*

Given a set of safety properties $\{\mathcal{P}\}_{i=1}^n$ and a candidate DNN f_θ, we define the DNN Repair problem as the problem of repairing the DNN to generate a new DNN f'_θ such that all the properties are satisfied, as defined in Problem 1. While repairing DNNs, it is also extremely important to preserve the parameter θ of the candidate DNN to the most extent because a subtle deviation on the parameter can lead to a high impact on the original performance and even introduce unexpected unsafe behaviors. These changes can be difficult to identify due to the black-box nature of DNNs. Therefore, our work also considers the minimal repair, which is formally defined in Problem 2.

Problem 1 (DNN Repair). Given a DNN candidate f_θ and a set of safety properties $\{\mathcal{P}\}_{i=1}^n$, at least one of which is violated, the repair problem is to train a new parameter set θ' based on θ, such that $f_{\theta'}$ satisfies all the safety properties, $f_{\theta'} \models \{\mathcal{P}\}_{i=1}^n$.

Problem 2 (Minimal DNN Repair). Given a DNN candidate f_θ and a set of safety properties $\{\mathcal{P}\}_{i=1}^n$, at least one of which is violated, the minimal repair problem is to train a new parameter set θ' based on θ, such that $f_{\theta'}$ satisfies all the safety properties, $f_{\theta'} \models \{\mathcal{P}\}_{i=1}^n$ while minimizing the L-distance $\|\theta' - \theta\|_L$.

For DNN classifier, we use classification accuracy on finite test data to analyze the performance of a repaired DNN $f_{\theta'}$ with respect to its original DNN f_θ. While for DNN regressor, we use the prediction error on test data. The parameter deviation $\|\theta' - \theta\|_L$ is used to evaluate the DNN change caused by the repair. Additionally, we also analyze the impact of the DNN deviation on the reachability of a DNN. Here, reachability indicates the reachable domain on safety properties. As reachability characterizes the behaviors of a DNN, a desired repair method should preserve its reachability. In the repair of DNN agents in DRL, the DNN performance is set to the averaged rewards on a certain number of episode tests.

2.2 DNN Agent Repair for Deep Reinforcement Learning

In Deep Reinforcement Learning (DRL), an agent is replaced with a DNN controller. The inputs to the DNN are states or observations, and their outputs correspond to agent actions. Similarly, a *property* $\mathcal{P}$ for the DNN agent defines a scenario where it specifies an input state space $\mathcal{I}$ containing all possible inputs, and also a domain $\mathcal{U}$ of undesired output actions. Different from regular DNN learning that uses existing training data, DRL learns through trial and error from their own experience collected from interactions with the environment. How to utilize the unsafe state domain computed with the reachability analysis to repair unsafe behaviors of the agent remains a problem. This problem is formally defined as follows:

Problem 3 (DNN Agent Repair). Given a DNN agent candidate f_θ and a set of safety properties $\{\mathcal{P}\}_{i=1}^{n}$, at least one of which is violated. The repair problem is to learn a DNN f'_θ through trial and error from the experience in the interactive environment, such that $f'_\theta \models \{\mathcal{P}\}_{i=1}^{n}$ while maximizing the reward.

2.3 Computation of Exact Unsafe Domain of ReLU DNNs

The exact unsafe reachable domain of a DNN is defined with respect to input-output safety properties. This domain is computed not only for safety verification but also for retraining purposes. The problem of computing the unsafe domain is known to be an NP-complete problem [5]. In this paper, the computation of the unsafe domain is based on [18]. In the following, we provide a brief overview of the algorithm.

Given a DNN f_θ and a safety property $\mathcal{P}$ that specifies an input domain $\mathcal{I}$ and an unsafe output domain $\mathcal{U}$, the reachability analysis algorithm computes a set of output reachable sets $\mathcal{S}_k$, the union of which is the exact output reachable domain $\mathcal{O}{=}\bigcup_{k=1}^{m}\mathcal{S}_k$. Here, a set $\mathcal{S}_k$ refers to a convex set. The computation is denoted as $\mathcal{O}{=}\mathbb{N}(\mathcal{I})$. This process is illustrated in Fig. 1. For each $\mathcal{S}_k$, we compute its overlap $\mathcal{S}_u^{[k]}$ with the specified unsafe output domain $\mathcal{U}$, and then apply a backtracking algorithm to compute its corresponding unsafe input subspace $\mathcal{E}_u^{[k]}$ which is also a convex set. The union of $\mathcal{S}_u^{[k]}$ is the exact unsafe output reachable domain $\mathcal{O}_u{=}\bigcup_{k=1}^{m}\mathcal{S}_u^{[k]}$ and the union of $\mathcal{E}_u^{[k]}$ is the exact unsafe input space $\mathcal{I}_u{=}\bigcup_{k=1}^{m}\mathcal{E}_u^{[k]}$. The backtracking process is denoted as $\mathcal{I}_u{=}\mathbb{B}(\mathcal{O}_u)$ in Fig. 1.

In the reachability analysis method described above, the set representation for $\mathcal{E}_u$ and $\mathcal{S}_u$ includes their vertices. These vertices of all $\mathcal{E}_u$s and $\mathcal{S}_u$s distribute over the entire unsafe input domain $\mathcal{I}_u$ and unsafe output reachable domain $\mathcal{O}_u$, respectively. In addition, $\mathcal{E}_u$ is actually a subset of a *linear region* of the DNN and the *linear region* is a maximal convex subset of the input domain to the DNN, over which the DNN is linear. Therefore, $\mathcal{S}_u$ and $\mathcal{E}_u$ have an affine mapping relation, and so do their vertices. Overall, these vertices can be utilized to approximate the distance between unsafe domains and safe domains in Equ. 3 for the repair process.

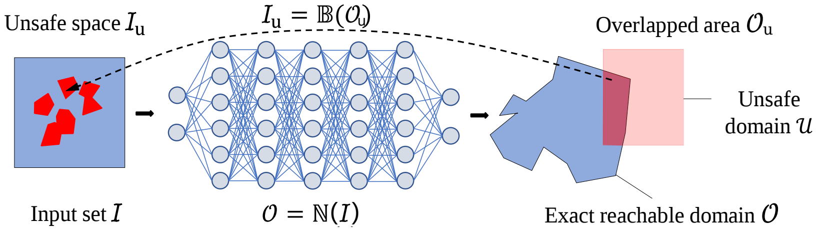

Unsafe space I_u
$I_u = \mathbb{B}(\mathcal{O}_u)$
Overlapped area $\mathcal{O}_u$
Unsafe domain $\mathcal{U}$
Input set I
$\mathcal{O} = \mathbb{N}(I)$
Exact reachable domain $\mathcal{O}$

Fig. 1: Computation of the unsafe input-output reachable domain $\mathcal{I}_u \times \mathcal{O}_u$.

3 Framework for DNN Repair

3.1 Formulation of Loss Function for Problems 1 & 2

We treat *Problem* 1 for DNN repair as a single-objective optimization problem where we seek to minimize the distance between the reachable unsafe domain and the known safe domain. For *Problem* 2, which defines the minimal DNN repair problem, we would also like to minimize the change in DNN parameters such that the repaired network does not falsify the specifications and its behavior is as close as possible to the original network.

Loss Function for DNN Repair We define a distance between the unsafe domain and the safe domain through the parameter set θ. By minimizing this distance, the unsafe domain can be gradually eliminated. The loss function can be formulated as:

$$\mathcal{L}_u(\theta) = \sum_{i=1}^{n} dist(\mathcal{O}_{\mathrm{u}}(\mathcal{I}^{[i]}, \mathcal{U}^{[i]}, \theta), \overline{\mathcal{U}^{[i]}}) \tag{1}$$

where $\mathcal{I}^{[i]}$ and $\mathcal{U}^{[i]}$ are the input domain and output unsafe domain specified by the property $\mathcal{P}_i$, the function *dist* computes the distance between the identified unsafe reachable domain $\mathcal{O}_u$ and the safe domain $\overline{\mathcal{U}}$. This distance is designed to be the minimum distance of each $\mathbf{y} \in \mathcal{O}_u$ to the safe domain $\overline{\mathcal{U}}$, such that the modification of unsafe behaviors can be minimal. This minimum l-norm distance of $\mathbf{y} \in \mathcal{O}_u$ to the safe domain can be formulated as $\min_{\hat{\mathbf{y}} \in \overline{\mathcal{U}}} \|\mathbf{y}(\mathbf{x}, \theta) - \hat{\mathbf{y}}\|_l$.

The common strategy of related works which aim to repair or improve the safety of DNNs with reachability analysis [17, 12, 8] is by considering the largest distance which is formulated as

$$dist: \quad \max_{\mathbf{y} \in \mathcal{O}_u} \min_{\hat{\mathbf{y}} \in \overline{\mathcal{U}}} \|\mathbf{y}(\mathbf{x}, \theta) - \hat{\mathbf{y}}\|_l. \tag{2}$$

However, the issues are twofold with this approach. First, it is unknown whether the unsafe domain $\mathcal{O}_u$ is convex or concave, which makes it significantly challenging to convert the computation of the exact largest distance to an LP problem. Secondly, Their solutions are based on the over-approximation of the output unsafe reachable domain by linearization of nonlinear activation functions. As a result, the approximation error is accumulated with neurons and it can be so conservative that a low-fidelity approximated distance may result in significant accuracy degradation. This remark is demonstrated in our experimental evaluation and comparison with the related work ART [8].

Different from these strategies, our reachability analysis method can obtain the exact unsafe reachable domain $\mathcal{O}_u$ efficiently. As introduced in Sec. 2.3, this is achieved by computing the exact unsafe *linear regions* $\{\mathcal{E}_u\}_{k=1}^m$ and their output reachable domains $\{\mathcal{S}_u\}_{k=1}^m$, where $\mathcal{S}_u^{[k]}{=}\mathbb{N}(\mathcal{E}_u^{[k]})$ and $\mathcal{O}_u{=}\bigcup_{k=1}^m \mathcal{S}_u^{[k]}$. The vertices of each pair of domains $\mathcal{E}_u^{[k]} \times \mathcal{S}_u^{[k]}$ can be denoted as $V_k : x \times y$. The $\mathbf{y} \in \mathcal{O}_u$ having the largest distance in Equ. 2 is also included in $\bigcup_{k=1}^m V_k$ which contains all the vertices of $\mathcal{O}_u$. Here, instead of identifying the $\mathbf{y}$, we choose to apply all the vertices to Equ. 1. This enables us to avoid searching the $\mathbf{y}$ in $\bigcup_{k=1}^m V_k$, which significantly reduces computation time. More importantly, since these vertices are distributed over the entire $\mathcal{O}_u$, they encode more geometrical information of this domain and hence are more representative than a single point $\mathbf{y}$ which only captures its largest distance to the safe domain $\overline{\mathcal{U}}$. Therefore, we substitute the *dist* function in Equ. 1 with a more general formulation:

$$dist: \quad \sum_{k=1}^{m} \sum_{j=1}^{|V_k|} \min_{\hat{\mathbf{y}} \in \overline{\mathcal{U}}} \|\mathbf{y}_j(\mathbf{x}_j, \theta) - \hat{\mathbf{y}}\|_l \tag{3}$$

where $|V_k|$ denotes the vertices set's cardinality.

In the following, we present our approach of approximating the closest safe $\hat{\mathbf{y}}$ to the unsafe $\mathbf{y}$. Recall that the unsafe domain $\mathcal{U}$ defined in the safety property is either a convex set formulated as $A\mathbf{x} + b \leq 0$ or a non-convex domain consisting of multiple such convex sets. Therefore, the problem of finding $\hat{\mathbf{y}}$ can be encoded as an LP problem of finding a $\hat{\mathbf{y}}$ on the boundaries of $\mathcal{U}$ such that the distance between $\hat{\mathbf{y}}$ and the interior $\mathbf{y}$ is minimal, where the optimal $\hat{\mathbf{y}}$ is located on one of its boundaries along its normal vector from $\mathbf{y}$. Let the vector from $\mathbf{y}$ to $\hat{\mathbf{y}}$ along the normal vector be denoted as $\Delta\mathbf{y}$. Then, the problem of finding $\hat{\mathbf{y}}$ can be formulated as

$$\hat{\mathbf{y}} = \mathbf{y} + (1+\alpha)\Delta\mathbf{y}, \quad \min_{\hat{\mathbf{y}} \notin \mathcal{U}} \|\mathbf{y} - \hat{\mathbf{y}}\| \tag{4}$$

where α is a very small positive scalar to divert $\hat{\mathbf{y}}$ from the boundary of $\mathcal{U}$ into the safe domain.

Loss Function for the Minimal DNN Repair The minimal repair problem is posed as a multi-objective optimization problem. In addition to the optimization for the repair problem explained previously, the minimal change of the DNN parameter θ is also considered in the problem formulation. For the minimization of the change, one simple and promising approach is to apply the training data in the retraining process of repair. Let the training input-output data be denoted as $\mathcal{X} \times \mathcal{T}$, then the function for measuring parameter change can be formulated

Algorithm 1 DNN Repair

```
Input: $\mathcal{N}$, $\{\mathcal{P}\}_{i=1}^{m}$, $(\mathbf{x}, \mathbf{y})_{training}$    ▷ an unsafe DNN, safety properties, training data
Output: $\mathcal{N}'$    ▷ an safe DNN satisfying all its safety properties.
procedure $\mathcal{N}'$ = Repair($\mathcal{N}$)
    $\mathcal{N}' \leftarrow \mathcal{N}$
    while $\mathcal{N}'$ is not safe on $\{\mathcal{P}\}_{i=1}^{m}$ do
        $\mathcal{D}_{unsafe}$ = reachAnalysis($\mathcal{N}$, $\{\mathcal{P}\}_{i=1}^{m}$)    ▷ compute unsafe data domains
        $\mathcal{L}_u$ = Dist($\mathcal{D}_{unsafe}$)    ▷ approximate the distance using Equ. 3.
        $\mathcal{L}_c$ = Loss($(\mathbf{x}, \mathbf{y})_{training}$)    ▷ compute loss on the training data in Equ. 5
        $\mathcal{N}'$ = Update($\mathcal{N}'$, $\mathcal{L}_u$, $\mathcal{L}_c$)    ▷ learn through the loss function in Equ. 6
```

as

$$\mathcal{L}_c(\theta) = \sum_{i=1}^{N} \|\mathbf{y}_i(\mathbf{x}_i, \theta) - \mathbf{t}_i\|_l \tag{5}$$

where $(\mathbf{x}, \mathbf{t}) \in \mathcal{X} \times \mathcal{T}$. Here, we combine the function $\mathcal{L}_u$ for repair in Equ. 3 and the function $\mathcal{L}_c$ in Equ. 5 into one composite loss function using the weighted sum, and the minimal repair process can be formulated as

$$\underset{\theta}{\text{minimize}} \Big(\alpha \cdot \mathcal{L}_u(\theta) + \beta \cdot \mathcal{L}_c(\theta) \Big) \tag{6}$$

where $\alpha, \beta \in [0, 1]$ and $\alpha + \beta = 1$. The configuration $\alpha = 1$, $\beta = 0$ indicates only the repair process, while $\alpha = 0$, $\beta = 1$ indicates the process does not include the repair but only the minimization of the parameter change.

The process of the DNN repair is described in Algorithm 1. Given an unsafe DNN candidate, in each iteration, its unsafe domain over safety properties are first computed. With the unsafe domain, the distance in Equ. 3 is then computed. Finally, together with the loss value on the training data, the total loss value in Equ. 6 is computed and used to updated the DNN parameters. The algorithm terminates when the DNN is verified safe or the maximum number of iterations is reached.

3.2 Repair for Deep Reinforcement Learning

DRL is a machine learning technique where a DNN agent learns in an interactive environment from its own experience. Our method aims to repair an agent which violates its safety properties while performance is maintained as defined in Problem 3. In each *time step* of DRL, the agent computes the action and the next state based on the current state. A reward is assigned to the state transition. This transition is denoted as a *tuple* $\langle s, a, r, s' \rangle$ where s is the current state, a is the action, s' is the next state, and r is the reward. Then, this tuple together with previous experience is used to update the agent. The sequence of *time steps* from the beginning with an initial state to the end of the task is called an *episode*. The DRL algorithm in this work considers one of the most popular algorithms, the deep deterministic policy gradients algorithm (DDPG) [7] and is utilized on the rocket-lander benchmark [5] inspired by the lunar lander [1].

Our repair method for DRL is demonstrated in Fig. 2. Given an unsafe agent candidate in Fig. 2(a), our reachability analysis method computes the unsafe state domain that leads to an unsafe action by the agent. The vertices of unsafe *linear region*s are selected as representative unsafe states for the unsafe domain. Instead of minimizing its distance to the closest safe state as proposed for the regular repair, we run one *episode* with the unsafe state as an initial state as shown in Fig. 2(b). In this process, a penalty r is applied to the unsafe action observed in the learning process, from which safety can be more naturally learned. The penalty r is normally set to the least reward in the old experience, where the *old experience* refers to the experience from learning the original unsafe agent. In the repair process, the *tuple* in each *time step* will be stored into a global buffer for previous experience, which is named *new experiences*. For training, a set of *tuple*s will be randomly selected from both experiences. The process in Fig. 2(a) will be repeated until the agent becomes safe. The process is also described in Algorithm 2.

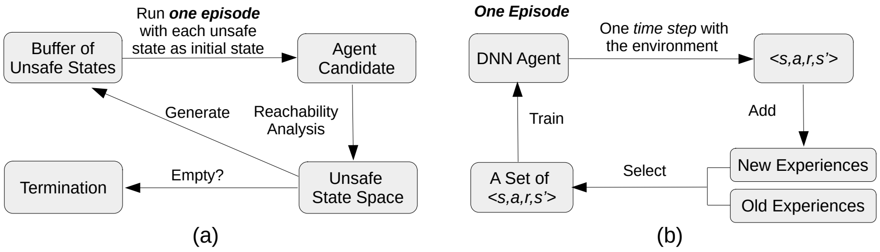


Fig. 2: Repair framework for deep reinforcement learning. In the loop (a), given an agent, its unsafe state space is first computed with our reachability analysis method. Then, *episode*s in (b) are run with unsafe states as initial states to update the agent, where the occurrence of unsafe states will be penalized.

4 Experiments and Evaluation

In this section, we evaluate our repair methods with two benchmarks. One is the DNN controllers for the Airborne Collision System X Unmanned (ACAS Xu) [4]. Our repair method in Section 3.1 is evaluated against the work ART [8]. We also study the different performance between our non-minimal repair method in Equ. 1 and our minimal repair in Equ. 5. To measure the impact of repair algorithms on DNNs, besides the accuracy on finite test data, we also analyze the changes of the DNN's reachability. The other benchmark is a set of DNN agents

[5] https://github.com/arex18/rocket-lander

Algorithm 2 Repair for Deep Reinforcement Learning

```
Input: $\mathcal{N}$, $E$, $\{\mathcal{P}\}_{i=1}^{m}$    ▷ an unsafe DNN agent, its old experience, safety properties
Output: $\mathcal{N}'$    ▷ a safe agent satisfying all its safety properties
procedure $\mathcal{N}' = $ Repair($\mathcal{N}$)
    $\mathcal{N}' \leftarrow \mathcal{N}$
    while $\mathcal{N}'$ is not safe on $\{\mathcal{P}\}_{i=1}^{m}$ do
        $\mathcal{D}_{unsafe} = $ reachAnalysis($\mathcal{N}$, $\{\mathcal{P}\}_{i=1}^{m}$)    ▷ compute unsafe state domains
        $S_{unsafe} = $ Vertices($\mathcal{D}_{unsafe}$)    ▷ representative unsafe states
        for $s$ in $S_{unsafe}$ do
            $\mathcal{N}' = $ Episode($\mathcal{N}'$, $s$, $E$)    ▷ one episode learning
```

for a rocket lander system based on the lunar lander [1]. With this benchmark, we explore our repair method proposed in Section 3.2 to repair unsafe DNN agents for DRL. The hardware for all experiments is Intel Core i9-10900K CPU @3.7GHz×, 10-core Processor, 128GB Memory, 64-bit Ubuntu 18.04.

4.1 Repair of ACAS Xu Neural Network Controllers

The ACAS Xu DNN controllers consist of an array of 45 fully-connected ReLU DNNs. They are used to approximate a large lookup table that converts sensor measurements into maneuver advisories in an airborne collision avoidance system, such that they can significantly reduce the massive memory usage and also the lookup time. All DNNs have the same architecture which includes 5 inputs, 5 outputs and 6 hidden layers with each containing 50 ReLU neurons. The 5 inputs correspond to the sensor measurement of the relative dynamics between the own ship and one intruder. The 5 outputs are prediction scores for 5 advisory actions. There are 10 safety properties defined, and each neural network is supposed to satisfy a subset of them.

Among these 45 network controllers, there are 35 unsafe networks violating at least one of the safety properties. Some works [8, 5] report there are 36 unsafe networks due to numerical rounding issues [16]. Since the original dataset is not publicly available, we uniformly sample a set of 10k training data and 5k test data from the state space of DNNs, the same strategy as ART [8]. To repair these unsafe networks, our minimal repair and ART [8] require these datasets while our non-minimal repair approach does not.

The parameter configurations for the retraining process of a DNN in our repair are as follows. For non-minimal repair, the learning rate $lr = 0.001$. The learning rate for our non-minimal repair normally needs to be small, because the retraining of the DNN is only guided by the modification of unsafe behaviors and a large value may also greatly affect other safe behaviors. For the minimal repair, which is a multi-objective optimization problem, a set of configurations are applied to estimate the optimal performance. Here, the learning rate lr and the value (α, β) in Equ. 6 are set as below. There are 6 different settings for the minimal repair of each unsafe network. The optimal result is selected for performance comparison. The loss functions in Equ. 1 and 5 are computed with the

	Learning Rate (lr)	(α, β)
Non-minimal Repair	0.001	-
Minimal Repair	{0.01, 0.001}	{(0.2,0.8),(0.5, 0.5),(0.8,0.2)}.

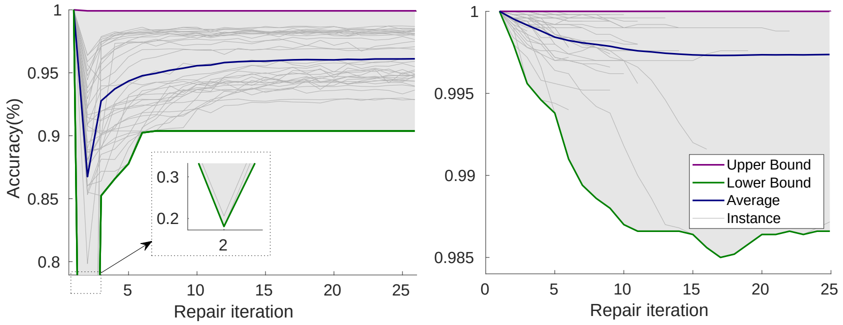
Euclidean norm. As introduced, each iteration of our repair consists of the reachability analysis and epochs of retraining. Here, we empirically set the maximum iteration to 100 and the number of epochs to 200 for all our repair methods. For ART [8], their default settings are applied for the comparison.

Success and Accuracy The experimental results are shown in Table 1. Table 1 describes the repair successes and the accuracy of repaired networks. Recall that the test data are sampled from the original network, therefore, the accuracy of the original network on these data is 100%. As shown, in terms of success, our non-minimal repair and minimal repair methods both successfully repair all 35 unsafe networks. ART can repair 33 networks. While ART with refinement which computes tighter approximation than ART can repair all of the networks. In terms of accuracy, our repaired networks exhibit a higher accuracy than ART and some of our repaired networks even have 100% accuracy, indicating less performance degradation. We hypothesize that the difference of performance in ART, as discussed in Section 1 is primarily due to the use of over-approximation methods for the unsafe domains of DNNs. They may be so conservative that the estimated distance in Equ. 1 is inaccurate, resulting in performance degradation. It can be also noticed that with the refinement in ART which computes a tighter approximation of domains, the number of repair successes and their accuracy increase. Overall, with the exact reachability analysis, our methods can outperform ART in terms of accuracy.

Table 1: Repair of ACAS Xu neural network controllers.

Methods	Repair Successes	Min Accu.	Mean Accu.	Max Accu.
Art	33/35	88.74%	94.87%	99.92%
Art-refinement	35/35	90.38%	96.23%	99.92%
Our Non-minimal Repair	**35**/35	**98.66**%	**99.74**%	**100.0**%
Our Minimal Repair	**35**/35	**99.38**%	**99.83**%	**100.0**%

For our non-minimal repair and minimal repair methods, we can notice that the accuracy difference of their repaired networks in Table 1 is trivial. Recall that our minimal repair can have the Pareto optimality issue in its multi-objective optimization in Equ. 6, while our non-minimal repair does not. It means that in the minimal repair, the minimization of the DNN deviation may impede the optimization for repair. By contrast, our non-minimal repair can consistently repair all networks with one parameter setting and meanwhile maintain the high accuracy without the usage of training data.



(a) ART with refinement

(b) Our method

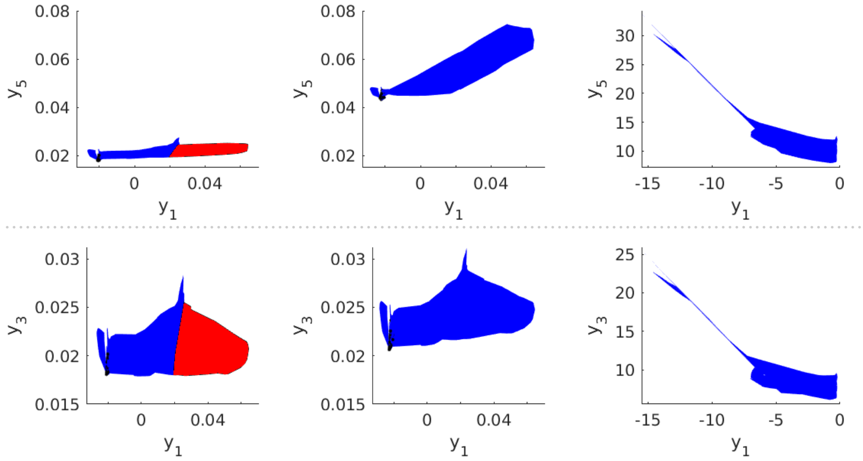
Fig. 3: Accuracy evolution of models with respect to the number of repair iterations. All models before the repair are unsafe on at least one safety property. All repairs successfully generate safe models in the end.

DNN Deviation after Repair Additionally, the accuracy evolution of networks under repair is also demonstrated in Fig. 3. It includes ART with refinement (a) and our non-minimal repair method (b). We can notice that at the beginning of ART, the accuracy of the repaired DNN will first drop quickly and in some instances, it even drops below 20%. Then, the accuracy gradually converges to a higher value. We speculate that at the beginning of the repair, ART mainly generates a safe model with a large modification of the original network and then, train this safe network with the training data to improve the accuracy.

We also analyze the impact of repair on the reachability of DNNs. The reachability refers to the output reachable domain of DNNs on the input domains $\mathcal{I}$ of their safety properties. Here, we consider the network N_{21} which includes safety properties 1,2,3,4. It violates Property 2 whose unsafe output domain is that $\mathbf{y}_1$ is the maximum. The output reachable domain of N_{21} has 5 dimensions, and it is projected on two dimensions for visualization.

The output reachable domains of N_{21} on Properties 1 & 2, projected on $(\mathbf{y}_1, \mathbf{y}_3)$ and $(\mathbf{y}_1, \mathbf{y}_5)$, are shown in Fig. 4. (a) represents the reachable domain of the original unsafe N_{21}. (b) and (c) represents the reachable domain of repaired N_{21} by our method and ART, respectively. The blue area represents the safe reachable domain, and the red area represents the unsafe reachable domain. We can notice that the unsafe domain is successfully eliminated by our method and ART. We can also notice that compared to ART, our method barely changes the reachable domain. With respect to the original reachable domain, our reachable domain on $(\mathbf{y}_1, \mathbf{y}_5)$ exhibits a more obvious change than the one on $(\mathbf{y}_1, \mathbf{y}_3)$. This is because in the majority of safety violations, $\mathbf{y}_5$ is the second-largest output, next to $\mathbf{y}_1$. Therefore, our repair modifies $\mathbf{y}_5$ the most to eliminate the unsafe domain, which avoids large changes on other dimensions.

In addition, the reachability on Properties 3 & 4, projected on $(\mathbf{y}_1, \mathbf{y}_5)$, is also shown in Fig. 5. Similarly, we can notice that the impact of our repair method



(a) The original (b) Our method (c) ART

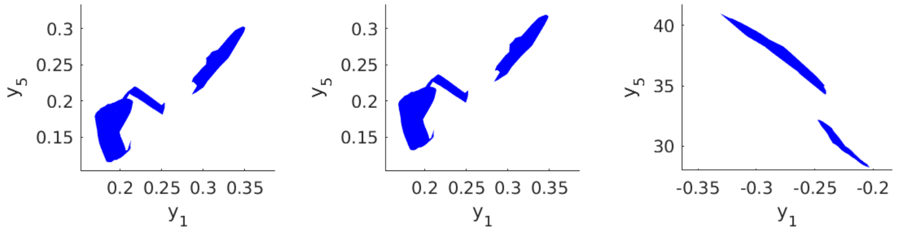
Fig. 4: Reachability of repaired network N_{21} on Properties 1 & 2, projected on dimensions $(\mathbf{y}_1, \mathbf{y}_5)$ and $(\mathbf{y}_1, \mathbf{y}_3)$. It shows the output reachable domains of the original network and its repaired networks. The red area represents the unsafe reachable domain, while the blue area represents the safe domain.

on the reachability of N_{21} is negligible, but ART changes the entire reachable domain. It can justify that a slight deviation on the DNN parameter may cause a tremendous change in its behaviors. It also shows that our DNN repair on one property hardly affects the DNN performance on other properties.

Table 2: Running time (**sec**) of our repair method and ART

Methods	**Regular Cases** (33 Nets)			**Hard Cases** (2 Nets)	
	Min	Mean Time	Max	Time (N_{19})	Time (N_{29})
Art	66.5	71.4	100.3	**65.6**	**71.5**
Art-refinement	85.4	89.2	90.1	84.6	90.4
Our Method	7.4	**65.5**	230.1	7173.2	3634.9

The running time of our repair method and ART is shown in Table 2. Here, we divide the repair of all 35 networks into regular cases and hard cases in terms of the volume of input domain of their safety properties. Normally, a larger input domain requires more computation for reachability analysis. The regular cases include all 33 networks whose safety properties specify small input domains. While the hard cases include the 2 networks N_{19} and N_{29} whose safety properties 7 and 8 specify large input domains. It can be noticed that our method is faster than ART in the regular cases but slower in the hard cases. That is



(a) The original (b) Our method (c) ART

Fig. 5: Reachability of repaired network N_{21} on Properties 3 & 4, on which the original network is safe. The domain is projected on dimensions $(\mathbf{y}_1, \mathbf{y}_5)$. We observe that due to the over approximation, ART repairs the network needlessly and changes the reachable set of the network.

because that unlike the over-approximation method utilized by ART, the exact reachability analysis of networks is an NP-complete problem [5]. When handling hard cases, it becomes less efficient. Despite this undesired efficiency in the hard cases, the exact analysis enables our method to repair all networks with much less performance degradation than ART.

4.2 Rocket Lander Benchmark

The rocket lander benchmark [6] is based on the lunar lander [1]. It is a vertical rocket landing model simulating SpaceX's Falcon 9 first stage rocket. Unlike the lunar lander whose action space is discrete, its action space is continuous, which commonly exists in the practical applications. Besides the rocket, a barge is also included on the sea which moves horizontally, and its dynamics are monitored. The rocket includes one main engine thruster at the bottom with an actuated joint and also two other side nitrogen thrusters attached to the sides of the top by unactuated joints. The main engine has a power F_E ranging in $[0, 1]$ and its angle relative to the rocket body is φ. The power F_S of the side thrusters ranges in $[-1, 1]$, where -1 indicates that the right thruster has full throttle and the left thruster is turned off, while 1 indicates the opposite. The rocket landing starts in certain height. Its goal is to land on the center of the barge without falling or crashing by controlling its velocity and lateral angle θ through the thrusters.

There are three actions, the main engine thruster F_E, its angle φ and the side nitrogen thrusters F_S. The observation contains the position x and y of the rocket relative to the barge, the velocity v_x and v_y of the rocket, its lateral angle θ, its angular velocity ω, and also last action advisory. It can be denoted as $[x, y, v_x, v_y, \theta, \omega, F'_E, \varphi', F'_S]$. Two safety properties are defined as below.

[6] https://github.com/arex18/rocket-lander.git

1. *Property 1*: for the state constraints $-20^\circ \leq \theta \leq -6^\circ$, $\omega < 0$, $\varphi' \leq 0^\circ$ and $F'_S \leq 0$, the desired action should be $\varphi < 0$ or $F_S < 0$, which prevents the rocket from tilting to the right in this state domain.
2. *Property 2*: for the state constraints $6^\circ \leq \theta \leq 20^\circ$, $\omega \geq 0$, $\varphi' \geq 0^\circ$ and $F'_S \geq 0$, the desired action should be $\varphi > 0$ or $F_S > 0$, which prevents the rocket from tilting to the left in this state domain.

The reinforcement learning algorithm Deep Deterministic Policy Gradients (DDPG) [7] is applied on this benchmark, which combines the Q-learning with Policy gradients. This algorithm is used for the environments with continuous action spaces. It consists of two models: Actor, a policy network that takes the state as input and outputs exact continuous actions, and Critic, a Q-value network that takes state and action as input and outputs Q-values. The Actor is our target agent controller. Here, among well-trained agents with DDPG, we first identify unsafe agents that violate these properties. Then, we apply our method to repair these agents. The Actor is designed with 9 inputs for state, 5 hidden layers with each containing 20 ReLU neurons, 3 outputs with subsequent *tanh* function.

Three unsafe agent controllers are learned. For the repair process, the learning rate for Actor and Critic is set to 10^{-4} and 10^{-3} respectively. The old experience from the learning process and the new experience from the current repair process are randomly selected for the learning, as shown in Fig. 2(b). A penalty reward is added for any wrong actions generated from input states. Its value is set to the lowest reward in the old experience. The change of performance is evaluated by $\mathbf{R} = (r' - r)/r$ where r' and r are the averaged reward of the repaired agent and the original agent tested on 1000 *episode*s.

Table 3: Repair of unsafe agents for the rocket lander. **ID** is the index of each repair. **R** denotes the performance change ratio of the repaired agent compared to the original unsafe agent. **Iter** denotes the number of iterations for repair. **Time** (*sec*) denotes the running time for one repair with our method.

	Agent 1			Agent 2			Agent 3		
ID	**R**	**Iter**	**Time**	**R**	**Iter**	**Time**	**R**	**Iter**	**Time**
1	+0.063	3	332.7	+0.048	3	635.7	+0.053	2	446.1
2	+0.088	3	302.0	+0.012	6	1308.4	+0.085	3	1451.6
3	+0.079	3	447.9	-0.084	4	812.9	-0.033	3	2417.1
4	+0.078	3	884.2	+0.025	3	620.3	+0.073	2	1395.3
5	+0.085	3	754.3	-0.001	4	813.5	-0.165	5	2632.9

For each unsafe agent, we conduct repair 5 times with each repair aiming to obtain a safe agent. There are totally 15 instances. The experimental results are shown in Table 3, which describe the performance change ratio **R**, the iterations of repair and the total time. We note that our framework can successfully repair the 3 agents in all 15 instances. In most cases, the performance of the repaired agent is slightly improved. The performance degradation in other instances is also trivial. The repair process takes 2-6 iterations for all instances with the running time ranging from 332.7 seconds to 2632.9 seconds. The evolution of the reachability of repaired network is also shown in Fig. 6. It shows that our repair only slightly affects the reachable domain of the agent.

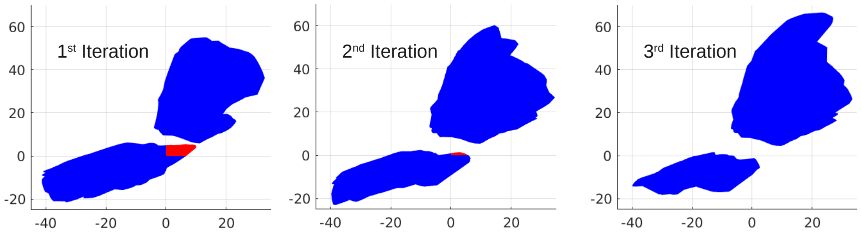


Fig. 6: The evolution of the output reachable domain on Property 1&2 in the repair of Agent 1 on ID 1. The domain is projected on $(\mathbf{y}_2, \mathbf{y}_3)$. The blue area represents the exact output reachable domain while the red area represents the unsafe reachable domain.

5 Conclusion and Future Work

We have presented methods to repair unsafe DNN controllers for autonomous systems. our method can be utilized to repair unsafe DNNs, even without training data. It can also be integrated into existing reinforcement algorithms to synthesize safe DNN controllers. Our experimental results on two practical benchmarks have shown that our method can successfully obtain a provably safe DNN while maintaining its accuracy and performance.

Acknowledgements

The material presented in this paper is based upon work supported by the National Science Foundation (NSF) through grant numbers 1910017 and 2028001, the Defense Advanced Research Projects Agency (DARPA) under contract number FA8750-18-C-0089, and the Air Force Office of Scientific Research (AFOSR) under contract number FA9550-22-1-0019. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of AFOSR, DARPA, or NSF.

References

1. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. arXiv preprint arXiv:1606.01540 (2016)
2. Goldberger, B., Katz, G., Adi, Y., Keshet, J.: Minimal modifications of deep neural networks using verification. In: LPAR. vol. 2020, p. 23rd (2020)
3. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
4. Julian, K.D., Lopez, J., Brush, J.S., Owen, M.P., Kochenderfer, M.J.: Policy compression for aircraft collision avoidance systems. In: 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC). pp. 1–10. IEEE (2016)
5. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient smt solver for verifying deep neural networks. In: International Conference on Computer Aided Verification. pp. 97–117. Springer (2017)
6. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., et al.: The marabou framework for verification and analysis of deep neural networks. In: International Conference on Computer Aided Verification. pp. 443–452. Springer (2019)
7. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
8. Lin, X., Zhu, H., Samanta, R., Jagannathan, S.: Art: Abstraction refinement-guided training for provably correct neural networks. In: FMCAD. pp. 148–157 (2020)
9. Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., Kochenderfer, M.J.: Algorithms for verifying deep neural networks. arXiv preprint arXiv:1903.06758 (2019)
10. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083 (2017)
11. Majd, K., Zhou, S., Amor, H.B., Fainekos, G., Sankaranarayanan, S.: Local repair of neural networks using optimization. arXiv preprint arXiv:2109.14041 (2021)
12. Mirman, M., Gehr, T., Vechev, M.: Differentiable abstract interpretation for provably robust neural networks. In: International Conference on Machine Learning. pp. 3578–3586 (2018)
13. Sohn, J., Kang, S., Yoo, S.: Search based repair of deep neural networks. arXiv preprint arXiv:1912.12463 (2019)
14. Tran, H.D., Yang, X., Lopez, D.M., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: International Conference on Computer Aided Verification. pp. 3–17. Springer (2020)
15. Usman, M., Gopinath, D., Sun, Y., Noller, Y., Pasareanu, C.: Nnrepair: Constraint-based repair of neural network classifiers. arXiv preprint arXiv:2103.12535 (2021)
16. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: 27th {USENIX} Security Symposium ({USENIX} Security 18). pp. 1599–1614 (2018)
17. Wong, E., Kolter, Z.: Provable defenses against adversarial examples via the convex outer adversarial polytope. In: International Conference on Machine Learning. pp. 5286–5295. PMLR (2018)
18. Yang, X., Johnson, T.T., Tran, H.D., Yamaguchi, T., Hoxha, B., Prokhorov, D.: Reachability analysis of deep relu neural networks using facet-vertex incidence. In: Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control. HSCC '21, Association for Computing Ma-

chinery, New York, NY, USA (2021). https://doi.org/10.1145/3447928.3456650, https://doi.org/10.1145/3447928.3456650

19. Yang, X., Yamaguchi, T., Tran, H.D., Hoxha, B., Johnson, T.T., Prokhorov, D.: Reachability analysis of convolutional neural networks. arXiv preprint arXiv:2106.12074 (2021)