9-15-2022

Neural Network Repair with Reachability Analysis

Xiaodong Yang

Tom Yamaguchi

Tran Hoang-Dung

Bardh Hoxha

Taylor T. Johnson

See next page for additional authors

Authors

Xiaodong Yang, Tom Yamaguchi, Tran Hoang-Dung, Bardh Hoxha, Taylor T. Johnson, and Danil Prokhorov

Published in LNCS 13465: Formal Modeling and Analysis of Timed Systems.
20th International Conference, FORMATS 2022. Warsaw, Poland, September 13–15, 2022,
Proceedings, pp. 221–236.

Neural Network Repair with Reachability Analysis

XIAODONG YANG, Vanderbilt University, USA
TOM YAMAGUCHI, TRINA, Toyota NA R&D, USA
HOANG-DUNG TRAN, University of Nebraska, USA
BARDH HOXHA, TRINA, Toyota NA R&D, USA
TAYLOR T JOHNSON, Vanderbilt University, USA
DANIL PROKHOROV, TRINA, Toyota NA R&D, USA

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. Formally verifying the safety and robustness of well-trained DNNs and learning-enabled cyber-physical systems (Le-CPS) under adversarial attacks, model uncertainties, and sensing errors is essential for safe autonomy. This research proposes a framework to repair unsafe DNNs in safety-critical systems with reachability analysis. The repair process is inspired by adversarial training which has demonstrated high effectiveness in improving the safety and robustness of DNNs. Different from traditional adversarial training approaches where adversarial examples are utilized from random attacks and may not be representative of all unsafe behaviors, our repair process uses reachability analysis to compute the exact unsafe regions and identify sufficiently representative examples to enhance the efficacy and efficiency of the adversarial training.

The performance of our repair framework is evaluated on two types of benchmarks without safe models as references. One is a DNN controller for aircraft collision avoidance with access to training data. The other is a rocket lander where our framework can be seamlessly integrated with the well-known deep deterministic policy gradient (DDPG) reinforcement learning algorithm. The experimental results show that our framework can successfully repair all instances on multiple safety specifications with negligible performance degradation. In addition, to increase the computational and memory efficiency of the reachability analysis algorithm in the framework, we propose a depth-first-search algorithm that combines an existing exact analysis method with an over-approximation approach based on a new set representation. Experimental results show that our method achieves a five-fold improvement in runtime and a two-fold improvement in memory usage compared to exact analysis.

Additional Key Words and Phrases: Neural network repair, reachability analysis, safe reinforcement learning

1 INTRODUCTION

Despite success of deep neural networks (DNNs) in various applications, trustworthiness is still one of the main issues preventing widespread use. Research has shown that DNNs may generate undesired behaviors even with the slightest perturbations on input data. Recently, many techniques for analyzing behaviors of DNNs have been presented [Anderson et al. 2020; Botoeva et al. 2020; Dutta et al. 2018; Frankle et al. 2020; Katz et al. 2019; Singh et al. 2019; Sotoudeh and Thakur 2021b; Tran et al. 2019c, 2020b; Urban et al. 2020; Urban and Miné 2021; Wang et al. 2020; Xiong and Jagannathan 2021; Yang et al. 2021a, 2020, 2021b]. Given a DNN, these works can generate a safety certificate over an input-output specification [Seshia et al. 2018]. However, due to the black-box nature of DNNs, training safe DNNs or repairing their erroneous behaviors remains a challenge.

Existing works to 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

Authors' addresses: Xiaodong Yang, Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN, USA, xiaodong.yang@vanderbilt.edu; Tom Yamaguchi, TRINA, Toyota NA R&D, Ann Arbor, MI, USA, tomoya.yamaguchi@toyota.com; Hoang-Dung Tran, University of Nebraska, Lincoln, NE, USA, trhoangdung@gmail.com; Bardh Hoxha, TRINA, Toyota NA R&D, Ann Arbor, MI, USA, bardh.hoxha@toyota.com; Taylor T Johnson, Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN, USA, taylor.johnson@vanderbilt.edu; Danil Prokhorov, TRINA, Toyota NA R&D, Ann Arbor, MI, USA, danil.prokhorov@toyota.com.

on neural weights that likely result in misbehavior. In [Sohn et al. 2019a], the authors propose a technique named *Arachne*. There, given a set of finite adversarial inputs that cause undesired behaviors, with guidance of a fitness function, *Arachne* searches and subsequently modifies neural weights that are likely related to these undesired behaviors. The method supports specifications consisting of a finite set of inputs instead of continuous regions. In [Goldberger et al. 2020], the authors propose 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 can be formally proved with the verification technique. However, the repair process is limited to a single adversarial example in each iteration. Typically, DNNs may contain multiple unsafe input regions over a continuous domain. In addition, the approach relies on modifications of the output layer, which may limit its capability. The second category utilizes adversarial examples for retraining. Adversarial training works such as [Goodfellow et al. 2014; Madry et al. 2017] have demonstrated that incorporating adversarial examples into the training process can improve the robustness of DNNs. However, DNNs may misbehave over continuous regions and infinitely many adversarial examples. Despite the robustness improvements, this training approach cannot guarantee safety for the learned DNNs. To solve this issue, some researchers incorporate reachability analysis in this process, such that they can train a model that is provably safe against norm-bounded adversarial attacks [Mirman et al. 2018; Wong and Kolter 2018]. Given a norm-bounded input range, these approaches over approximate the output reachable domain of DNNs with convex regions. Then they conduct robust optimization by minimizing the worst-case loss over these regions, which aims to migrate all the outputs to a 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 reachable domain can be so conservative that a low-fidelity worst-case loss may result in significant accuracy degradation. One promising alternative is to utilize exact reachability analysis methods [Bak et al. 2020; Tran et al. 2020a, 2019c, 2020b; Yang et al. 2021a]. These methods can compute the exact reachable set of DNNs and identify all the unsafe input regions.

In this paper, we propose a framework to repair DNNs, which combines adversarial training with exact reachability analysis of DNNs. We demonstrate the method and repair DNN controllers with respect to input-output safety specifications. In each iteration of the process, unsafe input regions are computed and incorporated into the training data. The iterative process will terminate once a model candidate is verified safe and also its performance is above a threshold. At the heart of our approach, we utilize a novel exact reachability method that is optimized for identification of the unsafe input regions. We also integrate our framework with learning algorithms of DNNs, specifically the deep reinforcement learning (DRL). The feasibility and effectiveness of this DNN repair framework is demonstrated on two types of benchmarks. One is an unsafe DNN of a horizontal collision avoidance system where the training data is accessible. For unsafe DNNs where the training data is available, the repair algorithm merges the unsafe regions of the model candidate to the training data in each loop, as shown in Figure 3. The other benchmark is an application of our framework on a DNN trained through a DRL algorithm. Here, the repair algorithm will be slightly modified since DRL is utilized to learn policies that maximize the expectation of rewards in the long term, as well as ensure reasonable behaviors by avoiding violations of safety constraints. The risk in DRL is normally associated with the inherent uncertainty of the environment and the facet that even an optimal agent may perform unsuccessfully with such stochastic natures. It is because the maximization of long-term rewards only involve finite environment and agent states, and it does not necessarily prevent rare occurrence of states that incur unsafe unsafe actions and subsequent safety violations. There is significant recent work on safe RL [Alshiekh et al. 2018; Bouton et al. 2019; Cheng et al. 2019; Fulton and Platzer 2018; Huang et al. 2020; Islam et al. 2020; Sohn et al. 2019b; Xiong et al. 2020; Zhou et al. 2020]. Most existing work relies on high-fidelity knowledge

of the environment dynamics and, to our best knowledge, there exist few approaches that can compute and eliminate risks from the environment uncertainty due to their non-determinism. In contrast, the advantages of our framework for DRL are threefold. Firstly, our framework can construct the all possible states as well as the uncertainties with regions. Secondly, our framework considers all unsafe state spaces in the regions and efficiently explore these spaces to reduce risks where the elimination of risks can be formally verified. Thirdly, our framework is well compatible with its learning process, with few adjustments needed for repair.

We summarize our contributions as follows: (1) We propose a framework for repair of DNNs with respect to input-output safety specifications. Our method does not require safe model references and can successfully repair unsafe DNNs on multiple safety specifications with negligible impact on performance. (2) The method can be utilized with deep reinforcement learning to generate provably safe agents. (3) We present a novel depth-first-search reachability analysis algorithm that includes both exact and over-approximation methods. This results in a five-fold computational speedup and two-fold memory reduction when compared to other state-of-the-art approaches. (4) The framework is evaluated on two benchmark problems where the detailed evolution of model candidates under repair is thoroughly analyzed.

2 PRELIMINARIES

2.1 Reachability Analysis and Set Representation

Reachability analysis is a process of computing reachable sets for the states of a system w.r.t. an initial state domain. For DNNs, given an input set bounding all possible inputs, reachability analysis computes its output reachable domain. In other words, it computes the domain of all possible outputs that the DNN can produce given an input range. The set normally refers to a convex polytope or a convex region bounded with linear constraints. In this process, sets will be sequentially updated by the affine mapping and activation functions in neurons in the DNNs until the last layer where the final sets compose the reachable domain of the DNN. The choice of set representation is a critical component of reachability analysis algorithms, and it has implications in computational complexity and accuracy of the approach. There are many mathematical structures that enable the definition of a convex polytope. For example, the *half-space representation* defines a polytope as a set of finite linear constraints. The *vertex representation* defines a polytope with a finite number of extreme points. The reachability analysis method in this work mainly relies on two set representations. One is the FVIM [Yang et al. 2021a] for exact reachability analysis, and the other one, a novel over-approximation approach proposed in this work, is the $\mathcal{V}$-zono representation proposed in Section 5. The new representation improves the computation speed and memory footprint of the algorithm. In the following, we review the FVIM representation for exact reachability analysis of DNNs [Yang et al. 2021a].

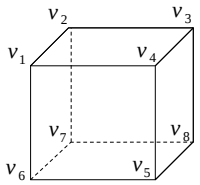
2.2 Facet-vertex Incidence Matrix

A facet-vertex incidence matrix (FVIM) is a complete encoding of the combinatorial structure of a convex set or a polytope [Henk et al. 2004]. It describes the containment relation between facets of a polytope and its vertices, where facets and vertices are types of faces and they are defined in Definition 2.1. The FVIM approach for exact reachability analysis of ReLU DNNs has been shown to be very efficient compared to approaches with different set representations [Yang et al. 2021a]. One example of the FVIM representation of a 3-dimensional polytope S is shown in Figure 6. The polytope contains eight vertices denoted as $\mathbf{v}$s and six facets denoted as Fs. Each facet is a *face* of S which contains four vertices. For instance, the facet F_1 denotes the plane containing vertices $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$ and $\mathbf{v}_4$. The complete containment relation between vertices $\mathbf{v}$s and facets Fs is encoded in

the FVIM on the left matrix. Together with real values of vertices, FVIM can represent the set S. Formally, it is defined as follows.

Definition 2.1 (Faces). *Given a polytope S and a supporting hyperplane $\mathcal{H} : a^\top x + b = 0$ whose halfpsace $a^\top x + b \leq 0$ or $a^\top x + b \geq 0$ contains S, if the dimension of F=$\mathcal{H} \cap S$ is k, then F is a k-dimensional face of S and denoted as k-face. A full-dimensional convex polytope $S \subseteq R^d$ contain 0-faces,1-faces, ..., (d-1)-faces which are respectively named vertex, edges, ..., facets. The cardinality of k-faces is denoted as $f_k(S)$.*

Definition 2.2 (Facet-vertex incidence matrix). *The facet-vertex incidence matrix of a full-dimensional polytope $S \in \mathbb{R}^d$ is a matrix $\mathcal{F} \in \{0,1\}^{f_{n-1}(S) \times f_0(S)}$ where the entry $\mathcal{F}(F, \boldsymbol{v})$=1 indicates that the facet F contains the vertex $\boldsymbol{v}$, while the entry $\mathcal{F}(F, \boldsymbol{v})$=0, otherwise.*



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
F_1	1	1	1	1	0	0	0	0
F_2	0	0	1	1	1	0	0	1
F_3	0	0	0	0	1	1	1	1
F_4	1	1	0	0	0	1	1	0
F_5	0	1	1	0	0	0	1	1
F_6	1	0	0	1	1	1	0	0

Fig. 1. Example of the facet-vertex incidence matrix.

Reachability analysis of DNNs with FVIM consists of the sequential application of two main processes. One is affine mapping of the input set by the weights and bias for each layer. This is followed by the transformation operation on the input set through each neuron in the layer. For affine mapping, one useful attribute of FVIM is that it actually only changes the value of vertices and will preserve the FVIM, which can ensure an efficient computation. As the input set passes through neurons, our algorithm checks whether the input range spans over the two linearities of the ReLU function. This is done by computing the lower bound and upper bound of the range. If it spans both linearities, then subsets belonging to different linearities are processed separately. The computation of the lower bound and upper bound of a set is one of primary challenges in the reachability analysis of DNNs. In other works, this problem is commonly encoded with LP solvers [Bak et al. 2020; Tran et al. 2019c; Wong and Kolter 2018; Zhang et al. 2018], which normally deal with a large number of variables and may exhibit undesired efficiency. In contrast, FVIM encodes all the vertices of the set which can be directly used to determine the lower bound and upper bound of the set. Thus the LP problems can be avoided.

3 DEEP NEURAL NETWORK REPAIR

It has been shown that training of DNNs with adversarial examples is an effective way to improve its robustness with respect to safety [Athalye et al. 2018; Goodfellow et al. 2014; Madry et al. 2017; Tramer et al. 2020]. These methods utilize a relatively small number of adversarial examples to train more robust DNNs. However, for DNN applications in safety-critical systems, additional guarantees are necessary. It is important to go from robustness improvements to safety guarantees without sacrificing performance of the DNN.

3.1 Provably Safe DNNs

Let $\mathcal{N} : X \rightarrow Y$ where X and Y are the input and output space be a Deep Neural Network such that given an input $\mathbf{x} \in X$, produces an output $\mathbf{y} = \mathcal{N}(x) \in Y$. The safety verification problem of DNNs w.r.t. safety properties is formally defined as follows.

Definition 3.1 (Safety Property). *A safety property $\mathcal{P}$ of a DNN $\mathcal{N}$ specifies an input domain $\mathcal{I} \subseteq X$ and a corresponding unsafe output domain $\mathcal{U} \subseteq Y$.*

Definition 3.2 (DNN Safety Verification). *A DNN is safe on a property $\mathcal{P}$, or $\mathcal{N} \models \mathcal{P}$, if for any $\boldsymbol{x} \in \mathcal{I}$ and $\boldsymbol{y} = \mathcal{N}(\boldsymbol{x})$ then $\boldsymbol{y} \notin \mathcal{U}$. Otherwise, it is unsafe, or $\mathcal{N} \not\models \mathcal{P}$.*

Given a set of safety properties $\{\mathcal{P}\}_{i=1}^{n}$, a performance function $\mathcal{A}$, and a candidate DNN $\mathcal{N}$, we define the DNN Repair problem as the problem of retraining or repairing the DNN to generate a new DNN $\mathcal{N}'$ such that all the properties are satisfied and the accuracy or performance of the candidate DNN is maintained. For classification DNNs, the performance function $\mathcal{A}$ refers to the classification accuracy on test data. For DNN agents in DRL, $\mathcal{A}$ refers to the averaged rewards on certain number of episode tests.

Problem 3.1 (DNN Repair). *Given a DNN candidate $\mathcal{N}$, safety properties $\{\mathcal{P}\}_{i=1}^{n}$ and performance function $\mathcal{A}$, train a DNN $\mathcal{N}'$ such that $\mathcal{N}' \models \{\mathcal{P}\}_{i=1}^{n}$ and also $A(\mathcal{N}') - A(\mathcal{N}) \geq \varepsilon$. ε is a constant value used to set the performance threshold.*

3.2 Reachability Analysis of DNNs with Backtracking

At the core of our approach, a reachability analysis method is utilized to determine specification violations. While traditional reachability analysis of neural networks focuses on computing output reachable domain given an input domain, for neural network repair, it is just as important to backtrack the unsafe reachable domain to the corresponding unsafe input domain containing all adversarial examples. The input domain that generates unsafe behaviors is then used for the training/repair process of the DNN. The computation of the unsafe input domain is normally associated with the computation of its output reachable domain. The algorithm needs first to determine the overlap between the reachable domain and the predefined unsafe domain before backtracking the corresponding unsafe input space. The computation of output reachable domain as well as the subsequent computation of unsafe input spaces are defined in Definition 3.3 and 3.4. They are also illustrated in Figure 2. Given an input set $\mathcal{I}_{in}$, a square input domain in blue, the exact output reachable domain $\mathcal{O}$ can be computed by $\mathcal{O} = \mathbb{N}(\mathcal{I}_{in})$. When $\mathcal{O}$ overlaps with the unsafe domain $\mathcal{U}$ which is $\mathcal{O}_u = \mathcal{O} \cap \mathcal{U}$ and $\mathcal{O}_u \neq \emptyset$, we can compute the unsafe input space $\mathcal{I}_u$ in red area that only contains all the inputs leading to the safety violation.

Definition 3.3 (Output Reachable Domain). *Let the computation of reachable sets of a DNN $\mathcal{N}$ be denoted as $\mathbb{N}(\cdot)$. Given an input set $\mathcal{I}_{in}$ to $\mathcal{N}$, a set of output reachable sets $\{S\}_{k=1}^{n}$ of $\mathcal{N}$ can be computed as $\{S\}_{k=1}^{n} = \mathbb{N}(\mathcal{I}_{in})$. We define the output reachable domain as $\mathcal{O} = \bigcup_{k=1}^{n} S_k$.*

Definition 3.4 (Unsafe Input Domain). *Given an input set $\mathcal{I}_{in}$ to a DNN $\mathcal{N}$ and its output reachable domain $\mathcal{O}$ that contains reachable sets $\{S\}_{k=1}^{n}$, if $\mathcal{O}$ overlaps with the unsafe domain $\mathcal{U}$, then it is denoted as $\mathcal{O}_u = \mathcal{O} \cap \mathcal{U}$ and $\mathcal{O}_u$ is named unsafe output reachable domain. The computation of the unsafe input domain refers to the process of computing a set of input subsets $\{I\}_{i=1}^{m} \subset \mathcal{I}$ and $\mathcal{I}_u = \bigcup_{i=1}^{m} I_i$, such that $\forall \boldsymbol{x} \in \mathcal{I}_u$, its output $\boldsymbol{y} \in \mathcal{O}_u$ and also that $\forall \boldsymbol{x} \notin \mathcal{I}_u$, its output $\boldsymbol{y} \notin \mathcal{O}_u$. This process is denoted as $\mathcal{I}_u = \mathbb{B}(\mathcal{O})$.*

Next, the algorithm for the computation of reachable sets will be presented in detail. We assume that DNNs consists of one input layer, multiple hidden layers with ReLU neurons, and one output layer with identity neurons. Except for the input layer, the computation in each layer includes *affine mapping* by the weights and bias ahead and also the process with neurons. *affine mapping* is denoted as $\mathbb{T}(\cdot)$. The computation function for each neuron is denoted as $\mathbb{E}(\cdot)$. Given an incoming set $S \in \mathbb{R}^d$ to a layer of l neurons, S will be first mapped by $\mathbb{T}(\cdot)$ into $S' \in \mathbb{R}^l$ where the dimension x_i of $\mathbf{x} \in S'$ is the input of i^{th} neuron.

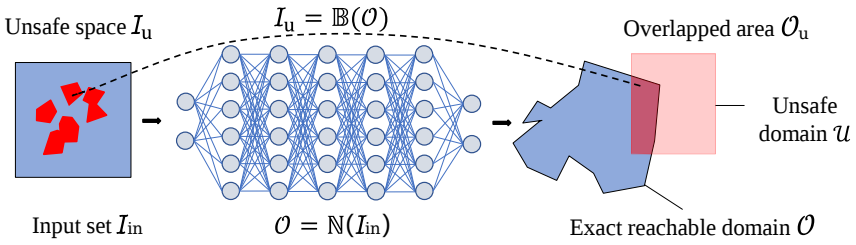


Fig. 2. Computation of exact reachable domain and identification of unsafe spaces.

Each ReLU neuron has two different linearities over its input range. For the i^{th} neuron, it can be denotes as $\hat{x}_i = \text{ReLU}(x_i)$ where for $x_i < 0$, $\hat{x}_i = 0$ and for $x_i \geq 0$, $\hat{x}_i = x_i$. For the process $\mathbb{E}_i(S')$ of S' with the ith neuron, there are totally three different cases. The first one is that S' only locates in the range $x_i < 0$. In this case, the dimension x_i of all $\mathbf{x} \in S'$ will be set to 0, which is equivalent to an *affine mapping* on S'. The second case is that S' only locates in the range $x_i \geq 0$, where S' will stay unchanged. The third case is that S' spans the two ranges. In this case, S' will be divided into two subsets by a hyperplane $\mathcal{H} : x_i = 0$, with each of subsets lying in one range $x_i < 0$ or $x_i \geq 0$. Then, on each subset, either the first or second case is applied.

Overall, the process for one input set with a neuron can generate at most 2 sets. These sets will be subsequently processed with another neuron until all the neurons in the layer are considered. Let the computation of one layer be denoted as $\mathbb{L}(\cdot)$. It then can be formulated as in Equation 1 where l denotes the number of neurons. The order in which the neurons in a layer are processed is not important. Given an input set S, in the worst case, it can output $O(2^l)$ sets. Based on Equation 1, the output reachable domain of a DNN can be computed layer by layer as in Equation 2 where k denotes the number of layers. Suppose the DNN includes n ReLU neurons, it will generate $O(2^n)$ reachable sets.

$$\mathbb{L}(S) = (\mathbb{E}_l \circ \cdots \circ \mathbb{E}_2 \circ \mathbb{E}_1 \circ \mathbb{T})(S) \tag{1}$$

$$\mathbb{N}(S) = (\mathbb{L}_k \circ \cdots \circ \mathbb{L}_2 \circ \mathbb{L}_1)(S) \tag{2}$$

Since different linearities of a ReLU neuron are separately considered in each $\mathbb{E}(\cdot)$, the computation of output reachable sets of a DNN is also equivalent to the reachability analysis for linear regions of the DNN. A *linear region* of a piecewise function like ReLU DNNs refers to the maximum convex subset of the input space, on which the function is linear. Taking this fact into account, the work [Yang et al. 2021a] proposes the set representation FVIM to track the connection between reachable sets and their linear regions, such that for any output sets that violates safe properties can be backtracked to its linear region and thus can identify the unsafe input space.

In order to compute the unsafe input domain, which is needed for DNN repair, we need to first compute the $O(2^n)$ reachable sets. In practice, only a portion of these reachable sets may violate safety specifications and a large amount of the computation is wasted on the safe reachable sets. Therefore, to improve the computational efficiency, we develop a method to filter out such sets and avoid additional computation.

Remark 3.1. *Our reachability analysis algorithm, in the worst case, will require computation of $O(2^n)$ reachable sets with n ReLU neurons in $\mathbb{N}(\cdot)$ and $\mathbb{B}(\cdot)$. We aim to develop an over-approximation method to verify the safety of sets computed in Equation 1, such that we can filter out the safe set that will not violate the properties $\mathcal{P}$s and avoid unnecessary subsequent computation.*

To solve this problem, we propose an algorithm that integrates an over-approximation method with the exact analysis method. Over-approximation methods can quickly check the safety of an

input set to DNNs. The integration is done as follows. Before an input set S is processed in a layer $\mathbb{L}(\cdot)$, its safety will be first verified with the over-approximation method. If it is safe, it will be discarded, otherwise, it continues with the exact reachability method. Suppose there are m ReLU neurons involved in the computation in Equation 2, then it can generate $O(2^m)$ reachable sets whose computation can be avoided if S is verified safe. The integration of the over-approximation algorithm also improves the memory footprint of the algorithm since a large number of sets are discarded early in the process. Another problem of the method based on Equation 1 and 2 is the memory-efficiency issue. As introduced, their computation may take up tremendous amount of the computational memory, may even result in out-of-memory issues, due to the exponential explore of sets. To solve this problem, the algorithm above is designed with the depth-first search, by which the memory usage can be largely reduced. The details of the over-approximation algorithm are presented in Section 5.3.

3.3 DNN 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. A *property* $\mathcal{P}$ for the DNN agent defines a scenario where it specifies an input state space $\mathcal{I}_{in}$ containing all possible inputs, and also a domain $\mathcal{U}$ of undesired output actions. Here, *safety* is associated with an input-output specification and reachability analysis refers to the process of determining whether a learned DNN agent violates any of its specifications and also the computation of unsafe state domain. This is formally defined as follows.

Definition 3.5 (Safe Agent). *Given multiple safety properties $\{\mathcal{P}_i\}_i^n$ for a DNN agent $\mathcal{N}$, the learned agent is safe if and only if for any $\mathcal{P}_i$, the reachable domain $\mathcal{O}^{[i]}$ for its input state space $\mathcal{I}_{in}^{[i]}$ by $\mathcal{O}^{[i]} = \mathbb{N}(\mathcal{I}_{in}^{[i]})$ does not overlap with its unsafe action space $\mathcal{U}^{[i]}$, namely, $\mathcal{O}^{[i]} \cap \mathcal{U}^{[i]} = \emptyset$.*

An unsafe agent has the state domains $\mathcal{O}_u$ where their states will result in unsafe actions. Since the traditional adversarial training is usually for regular DNN training algorithms with existing training data, how to utilize such states or adversarial examples in DRL to repair unsafe behaviors remains a problem. By considering the fact that DRL learns optimal policies in interactive environments by maximizing the expectation of rewards, one promising way will be introducing penalty to the occurrence of unsafe actions during the learning, such that safety can also be naturally learned from the unsafe state domain. This strategy can also ensure the compatibility with the DRL algorithms, such that they can be seamlessly integrated. Its details are presented in Section 4.1.

4 FRAMEWORK FOR DNN REPAIR

In this section, we propose a solution to Problem 3.1. The primary idea of our approach is to utilize reachability analysis to incorporate adversarial information into the training process. Different from regular adversarial training which obtains adversarial examples from random attacks, we consider the entire adversarial region by selecting representative examples which are sufficient to represent the region. The general approach is shown in Figure 3. The retraining process consist of several epochs. For each epoch, reachability analysis, as described in Figure 2, is conducted to compute the exact unsafe input domain $\mathcal{I}_u$ and its unsafe output reachable domain $\mathcal{O}_u$ for each safety property in $\{\mathcal{P}\}_{i=1}^n$. $\mathcal{I}_u$ and $\mathcal{O}_u$ together are named *unsafe data domain*, which is formally defined in Definition 4.1. Then data pairs $(\mathbf{x}, \mathbf{y})$ with $\mathbf{x} \in \mathcal{I}_u$ and $\mathbf{y} \in \mathcal{O}_u$ which sufficiently represent the unsafe input domain are selected. The selection process is presented next. After determining the representative data pairs, the unsafe output $\mathbf{y}$ for a particular adversarial example needs to be corrected before adding it to the original training data. This correction normally requires a safe

model as a reference. But in practice, this reference model is usually not available. Therefore, we propose two alternatives for the correction step. One is achieved by editing the unsafe $\mathbf{y}$ to its closest safe $\hat{\mathbf{y}}$ in the reachable space. The other is for the DRL where unsafe data pairs of the state and action will be penalized through rewards, such that safety can be naturally learned along with the optimal policies. The first case is presented next.

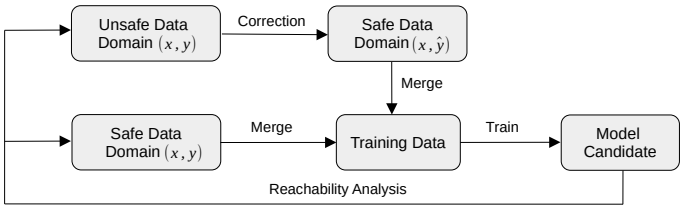


Fig. 3. Framework for neural network repair.

Definition 4.1 (Unsafe Data Domain and Unsafe Data Pair). *Given the unsafe input domain $\mathcal{I}_u$ of a DNN $\mathcal{N}$ on the safety property $\mathcal{P}$ with $\mathcal{I}_u = \bigcup_{k=1}^{m} I_k$, the unsafe reachable domain $\mathcal{O}_u$ is computed by $\mathcal{O}_u = \bigcup_{k}^{m} O_k$ where $O_k = \mathbb{N}(I_k)$ as described in Definition 3.4. Then the pair $\mathcal{I}_u$ and $\mathcal{O}_u$ is defined as the unsafe data domain of the DNN on the property $\mathcal{P}$, containing unsafe data pairs $(\boldsymbol{x}, \boldsymbol{y})$ with $\boldsymbol{x} \in \mathcal{I}_u$, $\boldsymbol{y} = \mathcal{N}(\boldsymbol{x}) \in \mathcal{O}_u$.*

Given a DNN with n safety properties and a set of l training data pair $(\mathbf{x}, \mathbf{y})$s, the DNN repair problem to satisfy all n properties as well as maintain its performance is formulated as

$$\underset{\theta}{\text{minimize}} \Big(\sum_{i=1}^{n} \max_{\mathbf{x} \in \mathcal{I}_u^{[i]}} L(f_\theta(\mathbf{x}), \hat{\mathbf{y}}) + \sum_{j=1}^{l} L(f_\theta(\mathbf{x}_j), \mathbf{y}_j) \Big) \tag{3}$$

where f denotes DNN, θ denotes the weight parameters, and $\hat{\mathbf{y}}$ represents the optimal safe output for the correction of $\mathbf{y}$ in the unsafe data pair $(\mathbf{x}, \mathbf{y})$ on property $\mathcal{P}_i$, and this correction refers to the **correction** procedure in Figure 3. Without safe model references for repair, we set $\hat{\mathbf{y}}$ to be the closest safe data to $\mathbf{y}$ in the space, on which $\|\mathbf{y} - \hat{\mathbf{y}}\|$ is minimal. The problem of finding $\hat{\mathbf{y}}$ can be encoded as a LP problem of finding a $\hat{\mathbf{y}}$ on the boundaries of $\mathcal{U}^{[i]}$ such that the distance between $\hat{\mathbf{y}}$ and $\mathbf{y}$ 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

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

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

The $\mathbf{y} \in \mathcal{O}_u^{[i]}$ that leads to the maximum loss value for the interior maximization of Equation 3 is from the extreme points of $\mathcal{O}_u^{[i]}$, namely, its vertices, because this loss is associated with the maximum distance $\Delta\mathbf{y}$ among $\mathbf{y} \in \mathcal{O}_u^{[i]}$. Let V_S be the set of vertices of $\mathcal{O}_u^{[i]}$, and V_k be the set of vertices of O_k where $O_k = \mathbb{N}(I_k)$. Since $\mathcal{O}_u^{[i]} = \bigcup_{k=1}^{m} O_k$ then $V_S \subseteq \bigcup_{k=1}^{m} V_k$. Recall that an unsafe input set I_k is a *linear region* of the DNN, over which the DNN is linear. Therefore, O_k is essentially an affine mapping from I_k and the vertices of I_k one-to-one correspond to V_k of O_k. We can conclude that the vertices of unsafe input sets $\{I\}_{k=1}^{m}$ contain the optimal $\mathbf{x} \in \mathcal{I}_u^{[i]} = \bigcup_{k=1}^{m} I_k$ for the interior maximization of Equation 3. Moreover, vertices are sufficient to represent a convex domain. Therefore, the vertices of unsafe input sets $\{I\}_{k=1}^{m}$ can sufficiently represent the unsafe

input domain $\mathcal{I}_u^{[i]}$, and data pairs $(\mathbf{x}, \mathbf{y})$ where $\mathbf{x}$ belongs to the vertices of $\{I\}_{k=1}^m$ and $\mathbf{y}$ belongs to the vertices $\{O\}_{k=1}^m$ can represent the unsafe data domain. These data pairs will be selected to merge into the training data for the adversarial training, which is the **merge** procedure in Figure 3.

The framework is also described in Algorithm 1. To maintain the performance of the repaired DNN, a threshold is also included in Line 5. Function **reachAnalysis** is used to compute all the safe and unsafe data domains of a DNN on multiple safety properties. Lines 4 and 7 generate representative data pairs for the adversarial training in Line 10. Function **Correction** applies a correction on $\mathbf{y}$ corresponding to Equation 4.

Algorithm 1 DNN Repair

Input: $\mathcal{N}$, $(\mathbf{x}, \mathbf{y})_{training}$ # an unsafe DNN and its training data
Output: $\mathcal{N}'$ # an safe DNN satisfying all its safety properties with a desired performance

1: **procedure** $\mathcal{N}'$ = Repair($\mathcal{N}$)
2: $\mathcal{N}' \leftarrow \mathcal{N}$
3: **while** true **do**
4: $\mathcal{D}_{unsafe}$, $\mathcal{D}_{safe}$ = reachAnalysis($\mathcal{N}$, $\{\mathcal{P}\}_{i=1}^m$) # compute unsafe and safe data domains
5: **if** $\mathcal{D}_{unsafe}$ is *empty* and $\mathcal{A}(\mathcal{N}') - \mathcal{A}(\mathcal{N}) \geq \varepsilon$ **then** # $\mathcal{A}$: performance function
6: **break** and **return** $\mathcal{N}'$
7: $(\mathbf{x}, \mathbf{y})_{safe}$, $(\mathbf{x}, \mathbf{y})_{unsafe}$ = Vertices($\mathcal{D}_{unsafe}$, $\mathcal{D}_{safe}$) # representative data pairs
8: $(\mathbf{x}, \hat{\mathbf{y}})$ = Correction($(\mathbf{x}, \mathbf{y})_{unsafe}$)
9: merge $(\mathbf{x}, \hat{\mathbf{y}})$, $(\mathbf{x}, \mathbf{y})_{safe}$ to $(\mathbf{x}, \mathbf{y})_{training}$
10: $\mathcal{N}'$ = Update($\mathcal{N}'$, $(\mathbf{x}, \mathbf{y})_{training}$)

4.1 Framework for Deep Reinforcement Learning

DRL is a machine learning technique where a DNN agent learns in an interactive environment from its own experience. Our framework aims to repair an unsafe agent which violates its safety properties while performance is maintained. The difference of the framework for DRL with the general framework in Figure 3 is the correction of $\mathbf{y}$ in unsafe data pairs $(\mathbf{x}, \mathbf{y})$. The correction in this modified framework is achieved by introducing a penalty to the unsafe data pair observed in the learning process, from which safety can be learned. In the following, we introduce the repair framework for DRL.

In a regular learning process, in each *time step*, 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 step*s from the beginning with an initial state to the end of the task is called an *episode*. With appropriate parameters settings, the performance of an agent may gradually converge to the optimum after a number of episodes. A good learning also relies on effective policy-learning algorithms. The DRL approach in this work considers one of the most popular algorithms, the deep deterministic policy gradients algorithm (DDPG) [Lillicrap et al. 2015] and is utilized on the rocket-lander benchmark [1] inspired by the lunar lander [Brockman et al. 2016].

As introduced, the correction of unsafe data pairs $(\mathbf{x}, \mathbf{y})$ is achieved through self-learning by assigning penalty to unsafe behaviors. Here, $\mathbf{x}$ refers to the state input s to the DNN agent and $\mathbf{y}$ refers to its output action a. The new framework is shown in Figure 4. Similar to the general framework, given an unsafe agent candidate in Figure 4(a), our reachability analysis method

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

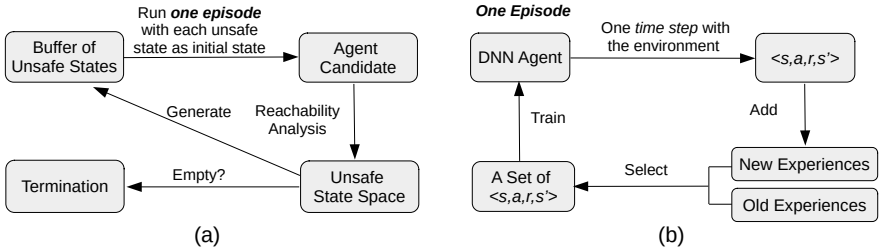


Fig. 4. Repair framework for deep reinforcement learning. The loop in (a) represents one epoch. Given an unsafe agent, its unsafe state space is computed with our reachability analysis method, Then, *eposides* are run with unsafe state as initial states to update the agent, where occurrence of unsafe states will be penalized. In (b), the new experiences refer to the experience learned during the repair while the old experiences refer to the ones learned in learning of the original agent.

computes the unsafe state domain that lead to a wrong action by the agent. The vertices of unsafe state sets $\{I\}_{k=1}^{m}$ are selected as representative unsafe states for the unsafe domain. The correction of the wrong action a for an unsafe state s will be achieved by running one *episode* with the unsafe state as an initial state as shown in Figure 4(b). The process of one *episode* is similar to the regular *episode*. The difference is that a new penalty r is incorporated for any unsafe pair s and a in each *time step*. The penalty r is normally being set to the least reward in the old experience, where the *old experiences* refers to the experience from learning the original unsafe agent. In the repair, 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 Figure 4(a) will be repeated until the agent becomes safe and its performance is above a predefined threshold. The algorithm is shown in Algorithm 2 where Function **singleEpisode** corresponds to Figure 2(b).

Algorithm 2 Repair for Deep Reinforcement Learning

Input: $\mathcal{N}$, E # an unsafe DNN agent, and its old experience, a set of *tuples*
Output: $\mathcal{N}'$ # a safe DNN satisfying all its safety properties with a desired performance

1: **procedure** $\mathcal{N}'$ = REPAIR($\mathcal{N}$)
2: $\mathcal{N}' \leftarrow \mathcal{N}$
3: **while** true **do**
4: $\mathcal{D}_{unsafe}$ = reachAnalysis($\mathcal{N}$, $\{\mathcal{P}\}_{i=1}^{m}$) # compute unsafe and safe data domains
5: **if** $\mathcal{D}_{unsafe}$ is *empty* and $\mathcal{A}(\mathcal{N}') - \mathcal{A}(\mathcal{N}) \geq \varepsilon$ **then** # $\mathcal{A}$: performance function
6: **break** and **return** $\mathcal{N}'$
7: S_{unsafe} = Vertices($\mathcal{D}_{unsafe}$) # representative unsafe states
8: **for** s in S_{unsafe} **do**
9: $\mathcal{N}'$ = singleEpisode($\mathcal{N}'$, s, E) # one episode learning with initial state s and E.

5 REACHABILITY ANALYSIS OF DNN

Fast reachability analysis is a core component in our DNN repair framework. However, different from traditional algorithms, for DNN repair the emphasis of the algorithm is on finding the unsafe input domain and it's corresponding unsafe output domain. Our algorithm builds on the reachability analysis and backtracking method presented in [Yang et al. 2021a]. The method utilizes

a FVIM set representation for efficient encoding of the combinatorial structure of polytopes. This set representation is suitable for set transformations that are induced by operations in a neural network. The reachability analysis method presented in [Yang et al. 2021a] is able to compute the output reachable domain of a DNN, and subsequently identify the unsafe input regions. However, one disadvantage of the algorithm is that, in the worst case, the number of reachable sets is $O(2^n)$, where n is the number of ReLU neurons. Its efficiency could be impeded due to this computation of a huge number of sets.

To alleviate this problem, we utilize an novel over-approximation method to speed up computation in Equation 1 and 2 by filtering out safe regions in the early stages of the algorithm. Since our focus of retraining is on computing the unsafe input domain and it's corresponding unsafe output domain, once we have guarantees of safety for a particular region, we do not need to compute its exact output reachable sets. Thereby, the computational efficiency and the memory footprint of the algorithm can be significantly improved.

Our over-approximation method is based on a new set representation for the linear relaxation of ReLU neurons. The new set representation named $\mathcal{V}$-zono is designed to efficiently encode the exponentially increasing vertices of sets in each linear relaxation, and it is totally compatible with the FVIM. In the following section, the over approximation with $\mathcal{V}$-zono will be introduced. Additionally, to handle the memory-efficiency issue caused by the large amount of sets computed in Equation 1 and 2, a depth-first search algorithm is also presented.

5.1 Over Approximation with Linear Relaxation

This section presents our over-approximation method based on the linear relaxation of ReLU. Linear relaxation is commonly used in other related works for fast safety verification of DNNs, such as [Gehr et al. 2018; Singh et al. 2019; Zhang et al. 2018]. Instead of considering the two different linearities of ReLU neuron over its input in $\mathbb{E}(\cdot)$ of Equation 1, these works apply one convex domain to over approximate these linearities to simplify the reachability analysis, as shown in Figure 5. This over approximation is named *linear relaxation*. Recall the process of S' with the ith neuron in the layer in Section 3.2, when the lower bound and the upper bound of the x_i of $\mathbf{x} \in S'$ spans the two linearities bounded by x_i=0, S' is supposed to be divided accordingly and their two subsets lying in the range $x_i < 0$ or $x_i \geq 0$ will be processed in terms of their linearity. The linear relaxation is applied only in such cases as shown in Figure 5 (b) and (c). When S' only locates in $x_i < 0$ or $x_i \geq 0$, the computation will be the same as the computation in Section 3.2. We denote this process including the linear relaxation as $\mathbb{E}^{app}(\cdot)$, and the computation of one layer as $\mathbb{L}^{app}(\cdot)$. Thus, by simply substituting $\mathbb{E}(\cdot)$ with $\mathbb{E}^{app}(\cdot)$ in Equation 1, and substituting $\mathbb{L}(\cdot)$ with $\mathbb{L}^{app}(\cdot)$ in Equation 2, we can conduct the over-approximation method shown in Equation 5 and 6. Function $\mathbb{E}^{app}(\cdot)$ only generates one output set for each input set instead of at most two sets. Therefore, given one input set to the DNN, Equation 6 computes one over-approximated output reachable set instead of $O(2^n)$ sets with n ReLU neurons.

$$\mathbb{L}^{app}(S) = (\mathbb{E}_l^{app} \circ \cdots \circ \mathbb{E}_2^{app} \circ \mathbb{E}_1^{app} \circ \mathbb{T})(S) \tag{5}$$

$$\mathbb{N}(S) = (\mathbb{L}_k^{app} \circ \cdots \circ \mathbb{L}_2^{app} \circ \mathbb{L}_1^{app})(S) \tag{6}$$

Here we introduce two of the most common types of linear relaxations for ReLU functions as shown in (b) and (c) of Figure 5. The linear relaxation in (b) uses the minimum convex bound. Compared to other linear relaxations, it can over approximate the output reachable domain with the least conservativeness. A less conservative relaxation typically leads to more accurate reachability analysis algorithms. The primary challenge for this relaxation is the estimation of the lower bound lb and the upper bound ub for each ReLU activation function. This is normally formulated as an LP

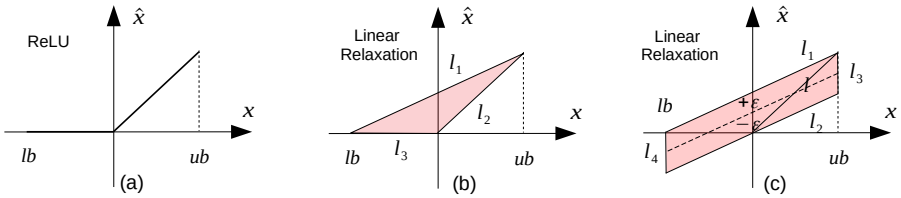


Fig. 5. Linear relaxations of ReLU functions with a convex bound: (a) ReLU function, (b) one type of linear relaxation of ReLU function [Wong and Kolter 2018], and (c) linear relaxation utilized in our over-approximation method based on a new set representation.

problem. However, since the number of variables equals to the number of activations, solving such problems with traditional methods for each verification may not be tractable.

One alternative to avoid the LP problems is to use vertices to represent a set, This also enables us to easily integrate this representation with the FVIM set representation in the exact reachability analysis. One issue with this approach is that doubling of vertices in each ReLU relaxation may add a significant computation cost and memory occupation. The explanation is as follows. Suppose the relaxation is for the ith neuron. As shown in (b) and (c), the relaxation introduces an unknown variable $\hat{x}_i$ to the incoming set $S \in \mathbb{R}^d$ and also the relation between x_i and $\hat{x}_i$ bounded in the convex domain.

Remark 5.1. *The introduction of $\hat{x}_i$ is equivalent to projecting S into S_h in $(d+1)$-dimensional space. For each $\boldsymbol{x} \in S$, it will transform into $\boldsymbol{x}_h \in S_h$ with $\boldsymbol{x}_h=[\boldsymbol{x}; \hat{x}_i] \in \mathbb{R}^{d+1}$. Accordingly, the faces of S will transform into new faces of S_h with increasing their dimension by one. The new faces of S_h are unbounded because of the unknown variable $\hat{x}_i$. The later intersection of S_h with the domain of x_i and $\hat{x}_i$ in the relaxation will yield real values to $\hat{x}_i$.*

For instance, the vertex $\mathbf{v}$ of S which is a 0-dimensional face will turn into $\mathbf{v}_h=[\mathbf{v}; \hat{x}_i]$ which is equivalent to an unbounded edge of S_h, a 1-dimensional face. With S_h and the linear relaxation bounds of x_i and $\hat{x}_i$, $\mathbb{E}_i^{app}(S)$ can be interpreted as the intersection of S_h with these bounds.

Remark 5.2. *The convex bounds of the ReLU relaxation consists of multiple linear constraint ls. Each $l : \alpha \cdot \boldsymbol{x} + \beta \leq 0$ is one of two halfspaces divided by the hyperplane $\mathcal{H} : \alpha \cdot \boldsymbol{x} + \beta = 0$. The intersection of S_h with each l is essentially identifying the subset of S_h which is generated from division of S_h by $\mathcal{H}$ and locates in the halfspace l.*

Take the (b) relaxation for instance which is bounded by three linear constraints l_1, l_2 and l_3. $\mathbb{E}_i^{app}(S)$ can be formulated as

$$\mathbb{E}_i^{app}(S) = S_h \cap \{\mathbf{x} \in \mathbb{R}^d \mid l_1 \cap l_2 \cap l_3\}. \tag{7}$$

As introduced above the vertex $\mathbf{v}_h=[\mathbf{v}; \hat{x}_i]$ of S_h is symbolic with $\hat{x}_i$ and is equivalent to an unbounded edge. In a bounded set, an edge includes two vertices. After the intersection of S_h with the linear constraints, $\mathbb{E}_i^{app}(S)$ generates a bounded subset of S_h and meanwhile, each symbolic $\mathbf{v}_h$ will yield to two real vertices. Therefore, the vertices of S_h is doubled from the vertices of S.

$$\begin{cases} \mathcal{H}_1 : (ub - lb) \cdot \hat{x} - ub \cdot x + ub \cdot lb = 0 \\ \mathcal{H}_2 : (ub - lb) \cdot \hat{x} - ub \cdot x = 0 \\ \mathcal{H}_3 : x - ub = 0 \\ \mathcal{H}_4 : x - lb = 0 \end{cases} \tag{8}$$

To solve this problem, it is necessary to develop a new set representation that can efficiently encode the exponential explosion of vertices with ReLU relaxations. Here, we choose the relaxation in Figure 5(c) because the convex bound for the relaxation is a *zonotope* which can be simply represented by a set of finite vectors and is formulated as a *Minkowski sum*. An efficient representation of vertices can benefit from this simplification. The explanation is as follows. The zonotope in (c) is bounded by two pairs of parallel supporting hyperplanes, $\mathcal{H}_1, \mathcal{H}_2$ and $\mathcal{H}_3, \mathcal{H}_4$ as shown in Equation 8, and their linear constraints are denoted as l_1, l_2, l_3 and l_4. Since l_3 and l_4 are lower and upper bounds of the x_i in S and S itself locates in these constraints, then, the left part of the conjunction in Equation 7 can be replaced with $\{\mathbf{x} \in \mathbb{R}^n \mid l_1 \cap l_2\}$. For each symbolic vertex $\mathbf{v}_h$=$[\mathbf{v}; \hat{x}_i]$, two new vertices $\mathbf{v}'_1$ and $\mathbf{v}'_2$ can be computed from the intersection of the hyperplanes $\mathcal{H}_1$ and $\mathcal{H}_2$ with $\hat{x}_i$ involved. The vertices $\mathbf{v}'_1$ and $\mathbf{v}'_2$ are shown in Equation 9 where x_i equals to the v_i of $\mathbf{v}$.

$$\mathbf{v}'_1 = \begin{bmatrix} \mathbf{v} \\ \frac{ub \cdot x_i - ub \cdot lb}{ub - lb} \end{bmatrix}, \quad \mathbf{v}'_2 = \begin{bmatrix} \mathbf{v} \\ \frac{ub \cdot x_i}{ub - lb} \end{bmatrix} \tag{9}$$

$$\{\mathbf{v}'_1, \mathbf{v}'_2\} = \left\{ \mathbf{v}'_c \pm \mathbf{v}'_v, \;\middle|\; \mathbf{v}'_c = \begin{bmatrix} \mathbf{v} \\ \frac{ub \cdot v_i}{ub - lb} - \frac{ub \cdot lb}{2(ub - lb)} \end{bmatrix}, \quad \mathbf{v}'_v = \begin{bmatrix} \mathbf{0} \\ \frac{ub \cdot lb}{2(ub - lb)} \end{bmatrix} \right\} \tag{10}$$

These two vertices can also be represented by Equation 10 where $\mathbf{v}'_v$ is a constant vector for the ReLU relaxation $\mathbb{E}_i^{app}(S)$. Let the vertices of S be V and S'=$\mathbb{E}_i^{app}(S)$, then for each $\mathbf{v} \in V$ it yields one $\mathbf{v}'_c$ for the vertices V' of S'. Let the set of $\mathbf{v}'_c$ be denoted as V'_c, then V' can be represented as Equation 11 where the doubled vertices are represented by plus-minus with the vector $\mathbf{v}'_v$.

$$V' = V'_c \pm \mathbf{v}'_v \tag{11}$$

The relaxation is illustrated by in Figure 6. The layer includes 2 ReLU neurons. The input set S is 2-dimensional with $\mathbf{x}$=$[x_1, x_2]^\top \in S$, and its vertices V consists of $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$. Here, x_1, x_2 respectively corresponds to the input of the first neuron and the second neuron. In this example, the process of S w.r.t. the first neuron is demonstrated. We can notice that the lower bound and the upper bound of x_1 in S are lb= -1 and ub=1, which indicates that S spans the input range of ReLU function over which the function exhibits two different linearities. Therefore, the linear relaxation is applied in terms of the subfigure (b). As introduced above, there are four linear constraints l_1, l_2, l_3 and l_4 bounding this relaxation of x_1 and $\hat{x}_1$, whose hyperplanes can be computed by Equation 8.

The introduction of new variable $\hat{x}_1$ projects 2-dimensional S into 3-dimensional S_h with $\mathbf{x} \in S$ transforming into $\mathbf{x}_h$=$[\mathbf{x}; \hat{x}_1] \in S_h$. Accordingly, the vertices $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$ of S transform into new symbolic vertices $[-1, 2, \hat{x}_1]^\top$, $[-1, 0, \hat{x}_1]^\top$ and $[1, 0, \hat{x}_1]^\top$ as shown in the subfigure (c), which are equivalent to three unbounded edges of S_h. After the intersection of S_h with those linear constraints, we obtain the final over-approximated set S' represented by the red domain in the subfigure (d) with its 6 vertices represented by the red points. According to Equation 10 and 11, the vertices V' of S can be represented as

$$\begin{bmatrix} x_1 \\ x_2 \\ \hat{x}_1 \end{bmatrix} \in V', \quad V' = \left\{ \mathbf{v}'_c \pm \mathbf{v}'_v \;\middle|\; \mathbf{v}'_c \in \left\{ \begin{bmatrix} -1 \\ 2 \\ -0.25 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ -0.25 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0.75 \end{bmatrix} \right\}, \; \mathbf{v}'_v = \begin{bmatrix} 0 \\ 0 \\ 0.25 \end{bmatrix} \right\} \tag{12}$$

where $\mathbf{v}'_c$s are denoted as $\{\mathbf{v}'_{c1}, \mathbf{v}'_{c2}, \mathbf{v}'_{c2}\}$ and described by the dark points in (d).

5.2 Over approximation with $\mathcal{V}-$zono

In the previous section, the a new set representation is preliminarily derived for the linear relaxation in Figure 5(c). This section mainly presents the formal definition of the set representation $\mathcal{V}$-zono, and its utilization in the over approximation of DNNs in Equation 5. The utilization includes the

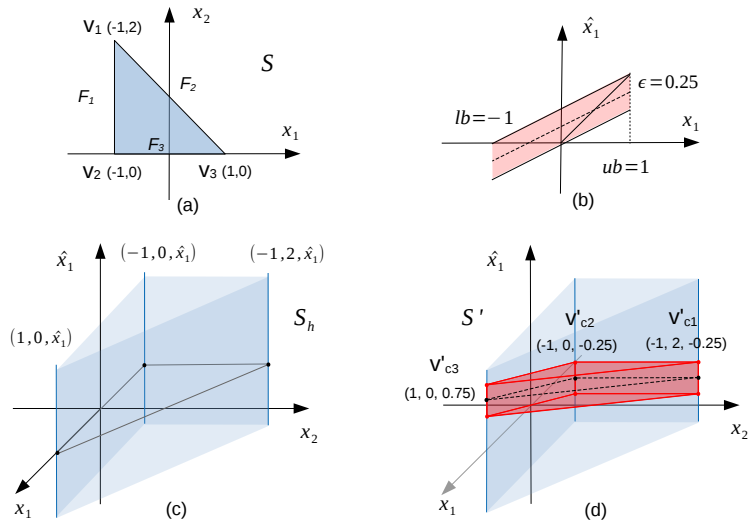


Fig. 6. Example of the linear relaxation.

linear relaxation of ReLU neuron $\mathbb{E}^{app}(\cdot)$, the affine mapping $\mathbb{T}(\cdot)$, as well as the safety verification on the safety properties of DNNs.

The $\mathcal{V}$-zono shares similarities with the $\mathcal{V}$-representation of polytopes introduced in Section 2, but it can efficiently encode exponentially increasing vertices. It is formally defined in Definition 5.1. It consists of *base vertices* C and *base vectors* $\mathcal{V}$. An example of the vertices representation is demonstrated in Figure 6(d). The convex set is the red domain. Its base vertices include three $\mathbf{v}'_c$ and the base vectors includes one $\mathbf{v}'_v$ as shown in Equation 12. Suppose C contains m base vertices and $\mathcal{V}$ contains n base vectors, then $C \pm \mathcal{V}$ efficiently represents $m \times 2^n$ vertices in Equation 13.

Definition 5.1 ($\mathcal{V}$-zono). *In the vertices representation, the set C that contains a set of finite real points $\boldsymbol{v}_c \in \mathbb{R}^d$ is named as Base Vertices, and the set $\mathcal{V}$ that contains a set of finite real vectors $\boldsymbol{v}_v \in \mathbb{R}^d$ is named Base Vectors. Then $\langle C, \mathcal{V} \rangle$ can represent a convex set $S \subset \mathbb{R}^d$ where $C \pm \mathcal{V}$ encodes all its vertices.*

$$C \pm \mathcal{V} = \{\boldsymbol{v}_c + \sum_{i=1}^{n} (\pm \boldsymbol{v}_{v,i}) \mid \boldsymbol{v}_c \in C \ \text{ and } \ \mathcal{V} = \{\boldsymbol{v}_{v,1}, \boldsymbol{v}_{v,2}, \ldots, \boldsymbol{v}_{v,n}\}\} \tag{13}$$

5.2.1 Linear Relaxation with $\mathcal{V}$-zono. In the application of $\mathcal{V}$-zono in ReLU relaxation, the vertex computation in $\mathbb{E}_i^{app}(S)$ has been formulated as Equation 10, which deals with regular vertices not represented by $\mathcal{V}$-zono. Here, we will formally present the linear relaxation with $\mathcal{V}$-zono where given an input set S in $\mathcal{V}$-zono, the $\mathcal{V}$-zono of $S' = \mathbb{E}_i^{app}(S)$ will be computed. Suppose the $\mathcal{V}$-zono of S has base vertices C and base vectors $\mathcal{V}$. In terms of Equation 10, a new base vertex $\mathbf{v}'_c$ can be computed for each $\mathbf{v}$ by

$$\mathbf{v}'_c = \begin{bmatrix} \mathbf{v} \\ \gamma \end{bmatrix}, \quad \gamma = \frac{ub \cdot v_i}{ub - lb} - \frac{ub \cdot lb}{2(ub - lb)}, \quad \mathbf{v} \in V \text{ and } V = C \pm \mathcal{V}.$$

Each $\mathbf{v}'_c$ is computed by incorporating one new dimension to each $\mathbf{v} \in V$ with a real value γ, which is essentially adding one new dimension to each $\mathbf{v}_c \in C$ with γ, and adding one new dimension to

each $\mathbf{v}_v \in \mathcal{V}$ with zero. Accordingly, all the new base vertices $\mathbf{v}'_c$s can be computed as

$$\left\{ \begin{bmatrix} \mathbf{v}_c \\ \gamma \end{bmatrix} + \sum_{i=1}^{n} \left(\pm \begin{bmatrix} \mathbf{v}_{v,i} \\ 0 \end{bmatrix} \right) \;\middle|\; \mathbf{v}_c \in C \text{ and } \mathbf{v}_{v,i} \in \mathcal{V} \right\}.$$

Based on the equation above, Equation 10 can be extended from the computation of new vertices $\mathbf{v}'$s for one $\mathbf{v}$ to all $\mathbf{v} \in V$. The vertices V' of S' can be computed as below, from which we can derive the C' and $\mathcal{V}'$ as shown in Equation 14.

$$\left\{ \begin{bmatrix} \mathbf{v}_c \\ \gamma \end{bmatrix} + \left(\sum_{i=1}^{n} \pm \begin{bmatrix} \mathbf{v}_{v,i} \\ 0 \end{bmatrix} \right) \pm \mathbf{v}'_v \;\middle|\; \mathbf{v}_c \in C, \text{ and } \mathbf{v}_{v,i} \in \mathcal{V} \right\}$$

$$C' = \left\{ \begin{bmatrix} \mathbf{v}_c \\ \gamma \end{bmatrix} \;\middle|\; \mathbf{v}_c \in C \right\}, \quad \mathcal{V}' = \left\{ \begin{bmatrix} \mathbf{v}_v \\ 0 \end{bmatrix}, \; \mathbf{v}'_v \;\middle|\; \mathbf{v}_v \in \mathcal{V} \right\} \tag{14}$$

From Equation 14, we notice that with the linear relaxation of each ReLU neuron $\mathbb{E}_i^{app}(\cdot)$, the dimension of vertices in $\mathcal{V}$-zono will be increased by one because by introducing the new dimension or variable $\hat{x}_i$ the old dimension x_i still remains. It will result in the dimension inconsistency with the subsequent affine mapping between layers. This old dimension can be eliminated with projecting the set S' on it by replacing the old dimension x_i in the vertices with the new $\hat{x}_i$. The projection is reflected on Equation 15 with the updated C' and $\mathcal{V}'$.

$$C' = \{\mathbf{v}_c \mid \forall \mathbf{v}_c \in C, \; \mathbf{v}_c[i] = \gamma\}, \quad \mathcal{V}' = \{\mathbf{v}_v, \; \mathbf{v}'_v \mid \forall \mathbf{v}_v \in \mathcal{V}, \; \mathbf{v}_v[i] = 0\} \tag{15}$$

After the projection, part of the points $C' \pm \mathcal{V}'$ will become the actual vertices of the projected set and the rest will become its interior points. The projection of a polytope S into a lower-dimensional space will generate another polytope S_l whose every face is a projection of a face of S. It indicates that the vertices V_l of S_l are from the projection of subset of the vertices V of S. Since after the projection Equation 15 preserves all projected vertices from Equation 14, there are redundant points in Equation 15 which are not vertices but only interior points of the projected polytope. This redundancy is allowed in the vertices representations of polytopes, as well as our set representation $\mathcal{V}$-zono. The detection and elimination of this redundancy requires and additional algorithm, which is out of the scope of this work and will be our future work. In addition, since $\mathcal{V}$-zono can efficiently encode vertices, the redundancy issue will not greatly affect the efficiency of the algorithm, which is also demonstrated in the experiments.

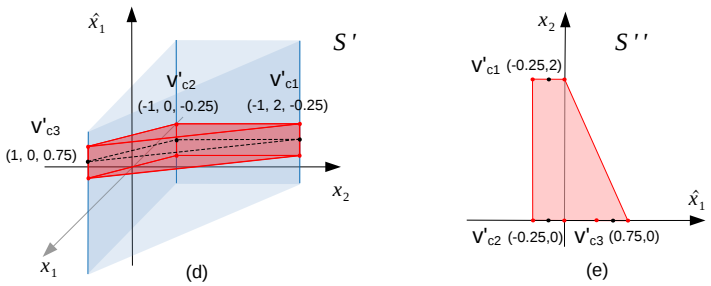


Fig. 7. Example of the linear relaxation.

An example of the projection of the set S' in Figure 5(d) is shown in Figure 7. In the ReLU relaxation, the x_1 is the old dimension and $\hat{x}_1$ is the new one, therefore, the set will be projected on

the dimension x_1 which will maintain the exact bounded relation between $\hat{x}_1$ and x_2. The projection generates another polytope represented by the red domain in (e). Its $\mathcal{V}$-zono for the vertices is

$$\begin{bmatrix}\hat{x}_1\\ x_2\end{bmatrix} \in V'', \quad V'' = \left\{ \mathbf{v}_c'' \pm \mathbf{v}_v'' \;\middle|\; \mathbf{v}_c'' \in \left\{ \begin{bmatrix}-0.25\\ 2\end{bmatrix}, \begin{bmatrix}-0.25\\ 0\end{bmatrix}, \begin{bmatrix}0.75\\ 0\end{bmatrix} \right\}, \; \mathbf{v}_v'' = \begin{bmatrix}0.25\\ 0\end{bmatrix} \right\}. \tag{16}$$

As shown in (d), its $\mathcal{V}$-zono contains 6 vertices denoted as red dots. After the projection as shown in (e), the new $\mathcal{V}$-zono contains 6 points which are the projections of these 6 vertices. 4 of them are actual the vertices of S'' and the rest 2 points are redundant.

5.2.2 Affine Mapping with $\mathcal{V}$-zono. As shown in Equation 5, the over approximation of reachable domain also includes affine mapping $\mathbb{T}_{(W,b)}(\cdot)$ between layers by weights W and bias b. As introduced in Section 2, affine mapping on a set S only changes the value of its vertices. Suppose the vertices V of S is represented by $C \pm \mathcal{V}$, then $\mathbb{T}_{(W,b)}(S)$ on V can be formulated as

$$\begin{aligned} W \cdot V + b &= W \cdot (C \pm \mathcal{V}) + b \\ &= W \cdot C + b \pm W \cdot \mathcal{V}. \end{aligned} \tag{17}$$

The new C', $\mathcal{V}'$ for new vertices V' after the affine mapping are

$$\begin{aligned} C' = W \cdot C + b &= \{W \cdot \mathbf{v}_c + b \mid \mathbf{v}_c \in C\} \\ \mathcal{V}' = W \cdot \mathcal{V} &= \{W \cdot \mathbf{v}_v \mid \mathbf{v}_v \in C\}. \end{aligned} \tag{18}$$

5.2.3 Safety Verification with $\mathcal{V}$-zono. The safety verification problem is to determine whether an output reachable domain overlaps with the unsafe domain bounded by linear constraints. Let one linear constraint be denoted as $l : \alpha^\top \cdot \mathbf{x} + \beta \leq 0$. Suppose vertices V of the over approximated output domain are represented by $C \pm \mathcal{V}$ and $\mathcal{V} = \{\mathbf{v}_{v,1}, \mathbf{v}_{v,2}, \ldots, \mathbf{v}_{v,n}\}$. Then, the verification of V w.r.t. the linear constraint l can transform into checking if the minimum value of $\alpha^\top \mathbf{v} + \beta$ over the vertices $\mathbf{v} \in V$ is not positive, which is formulated as

$$\text{minimize}(\alpha^\top (C \pm \mathcal{V}) + \beta). \tag{19}$$

The internal formula can be extended as

$$\alpha^\top (C \pm \mathcal{V}) + \beta = \alpha^\top C + \beta \pm \alpha^\top \mathcal{V} = \alpha^\top C + \beta + \sum_{i=1}^{k} \pm(\alpha^\top \mathbf{v}_v).$$

Then for each $\mathbf{v}_c \in C$, we can compute its minimum $\alpha^\top \mathbf{v}_c + \beta + \sum_{i=1}^{k} -|\alpha^\top \mathbf{v}_v|$. By computing the minimums of all $\mathbf{v}_c \in C$, we can determine the global minimum value in Equation 19 and thus complete the safety verification.

5.3 Fast Computation of Unsafe Input Spaces of DNNs

In the previous sections, an over-approximation method for the reachability analysis of DNNs based on the linear relaxation of ReLU neurons is developed. The method is formulated as Equation 5 and 6. As introduced, it is utilized to integrate with the exact analysis method formulated in Equation 1 and 2 to filter out all the safe intermediate sets that are computed in each $\mathbb{L}(\cdot)$ and are not necessary for the computation of unsafe input spaces for DNNs. In this section, the algorithm for such integration will be presented. This algorithm is based on the depth-first search to handle the memory-efficiency issue due to large amount of sets computed in each layer.

Algorithm 3 describes the integration of the computation of unsafe input spaces of DNNs with the proposed over-approximation method. Given an input domain S, it can compute all the unsafe input spaces w.r.t. the safety properties of the DNN. The details of each function are as follows.

(1) Function **Reach**$(\cdot)$ is a recursive function which, in each recursion, computes only one of input sets generated from the last layer to reduce the burden on the computational memory. Its base case is Line 3-5 where the computation reaches to the last layer of the DNN and unsafe input spaces will be computed. The recursion depth is the number of the DNN layers
(2) Function **outputOverApp**$(\cdot)$ over approximates the output domain of the DNN for each input set which is computed from the last layer with the exact analysis method in Equation 1 and 2. This function corresponds to Equation 5 and 6. The details is shown in Algorithm 4. In the beginning, the set S represented by FVIM is transformed into the $\mathcal{V}$-zono in Line 2. Subsequently, in each layer, it will be first processed by the affine mapping in Line 4 which corresponds to the $\mathbb{T}(\cdot)$ in Equation 5 and the computation of C' and $\mathcal{V}'$ in Equation 18. Then, in Line 5, it will be processed with ReLU neurons in the layer based on the linear relaxation. This process corresponds to the each $\mathbb{E}^{app}$ in Equation 5 and also the computation of C' and $\mathcal{V}'$ in Equation 15.
(3) Function **safetyCheck**$(\cdot)$ checks the safety of the over-approximated output domain computed in Line 6 with respect to safety properties. This function corresponds to Equation 19 where the vertices of the output domain are checked with each linear constraints of the unsafe domain defined in the properties. It returns *unsafe* if any vertex satisfies in all the constraints, otherwise, *safe*. When S is verified safe, the computation in the following layers can be abandoned because S will not lead to safety violation.
(4) Function **layerOutput**$(\cdot)$ computes the reachable sets for the current layer with input sets computed from the previous layer, which corresponds to Equation 1 and 2. All sets in this computation are represented by FVIMs. The details is shown in Algorithm 5.

The computational complexity of Algorithm 3 is that given an input set to a DNN with n ReLU neurons, $O(2^n)$ output reachable sets will be computed. In practice, the utilization of the over approximation method can significantly reduce the amount and improve the computational efficiency. The experimental results indicate that Algorithm 3 can be around five times faster than the algorithm without the over-approximation method.

Algorithm 3 Computation of unsafe input spaces of a neural network

Input: S # one input set to the neural network
Output: O_{unsafe} # O_{unsafe}: unsafe input spaces of the DNN

```
1:  procedure O_unsafe = REACH(S, layer)   # S: an input set; layer: the layer ID
2:      O_unsafe = empty
3:      if layer == lastlayer then   # lastlayer: the ID of the last layer
4:          unsafety = Backtrack(S)   # unsafety: unsafe input space for the unsafe domain in S
5:          return unsafety
6:      overapp = outputOverApp(S)   # overapp: over approximated output domain of the DNN
7:      if safetyCheck(overapp) then
8:          return None
9:      O_c = layerOutput(S, layer)   # O_c: output reachable sets of the current layer for S
10:     for S in O_c do
11:         O_unsafe.extend(Reach(S, layer+1))
12:     return O_unsafe
```

Algorithm 4 Reachable-domain Over approximation of a neural network

Input: S # one input set to the current layer
Output: O # one over approximated output reachable set of the current layer

```
procedure O = outputOverApp(S, layer)
    S = Vzono(S)    # transform the FVIM representation of S to the V-zono
    for layer = 1:lastlayer do
        S = affineMapping(S, layer)    # update base vertices and base vectors
        S = reluLayerRelaxation(S)    # relaxation of ReLU neurons
    return O ← S
```

Algorithm 5 Reachable set computation of one layer

Input: S # one input set to the current layer
Output: O # output reachable sets of the current layer

```
procedure O = layerOutput(S, layer)
    O == empty
    S = affineMapping(S, layer)
    if layer == lastlayer then
        return S
    O.extend(reluLayer(S))    # compute exact reachable sets with each ReLU neuron
    return O
```

6 EXPERIMENTS AND EVALUATION

In this section, we evaluates the performance of the framework, including the performance of the reachability analysis method. Recall from Section 4 that we consider two alternatives for the correction of unsafe data in the absence of a safe model reference. The first case considers transformations to the unsafe data points to the nearest safe set. The second case incorporates repair as part of the learning process. We evaluate the first case with a well-known benchmark named HorizontalCAS. HorizontalCAS is an airborne collision avoidance system, part of the ACAS Xu family proposed by [Julian and Kochenderfer 2019]. HorizontalCAS has neural networks as controllers. The code and training data for the benchmark are publicly available [2], based on which we train all DNNs. Unsafe DNNs will be first identified from these DNNs for the further repair with our framework. For the second approach, we apply our framework for repair in safe deep reinforcement learning on a well-known benchmark: the rocket lander based on the lunar lander [Brockman et al. 2016]. The hardware configuration is Intel Core i9-10900K CPU @3.7GHz×, 10-core and 20-thread Processor, 128GB Memory, 64-bit Ubuntu 18.04.

In addition to comparing with the related work [Yang et al. 2021a], our experimental evaluation examines whether the repairing process can converge on the multiple safety properties, how repairing unsafe behaviors on one property will affect other properties, and how the correction of unsafe data to its closest safe data affects the behavior of repaired DNNs.

6.1 HorizontalCAS DNN Controller Repair

The original controller for the HorizontalCAS is based on a Markov Decision Process (MDP) with large numeric tables [Julian and Kochenderfer 2019]. These controllers are replaced with neural networks. This reduces the memory footprint significantly. There are five continuous

[2]https://github.com/sisl/HorizontalCAS

inputs and two discrete inputs, For each combination of the two discrete input values, one neural network is trained. Overall, the controller consists of an array of 45 feed-forward neural networks. Each neural network is denoted as N_{ij} where i is an integer index ranging in $[1, 5]$ and j is an integer index ranging in $[1, 9]$. The input to each DNN is a 5 dimensional continuous sensor measurement of the dynamics between the ownship and the intruder. The inputs are denoted as $[\rho(\text{feet}), \theta(\text{deg}), \psi(\text{deg}), v_{\text{own}}(\text{feet/s}), v_{\text{int}}(\text{feet/s})]$ and they are, respectively, the distance between ownship and intruder, the angle of the ownship heading direction relative to intruder, angle of intruder heading direction relative to ownship heading direction, velocity of ownship and velocity of intruder. The lower bound and upper bound of their ranges is as follows:

$$lb = [0, -\pi, -\pi, 100, 0]; \quad ub = [56000, \pi, \pi, 1000, 1000].$$

There are 5 outputs corresponding to 5 action advisories which are, respectively, clear of conflict, weak right, strong right, weak left and strong left. The action with the maximum output will be selected. Each neural network includes 300 ReLU neurons which are fully connected. And for each neural network, there are several safety properties defined. In each safety property, for an input domain, desired action advisories are defined to avoid aircraft collision.

All 45 neural networks are well trained with the provided training data and default parameter settings in their code. The accuracy of the DNNs is over 94%. Here, we design three safety properties for all the networks in terms of the safety properties in work [Katz et al. 2017]. They are as follows:

(1) *Property 1*: for the input constraints $\rho \geq 50000$, $v_{\text{own}} \geq 900$ and $v_{\text{int}} \leq 60$, the desired output should be located in the domain where the output of the action advisory *clear-of-conflict* should not be the minimum. Then the unsafe output domain will be $y_1 \leq y_2 \cap y_1 \leq y_3 \cap y_1 \leq y_4 \cap y_1 \leq y_5$.
(2) *Property 2*: for the input constraints $1500 \leq \rho \leq 1800$, $-0.06 \leq \theta \leq 0.06$, $\psi \geq 3.10$, $v_{\text{own}} \geq 880$ and $v_{\text{int}} \geq 860$. The desired output should that the action advisory *clear-of-conflict* should not be the minimum.
(3) *Property 3*: for the input constraints $1500 \leq \rho \leq 1800$, $-0.06 \leq \theta \leq 0.06$, $\psi = 0$, $v_{\text{own}} \geq 900$ and $v_{\text{int}} \geq 700$. Their desired output is should that the action advisory *clear-of-conflict* should not be the minimum.

As introduced in Section 4, the correction of unsafe data pairs $(\mathbf{x}, \mathbf{y})$ is achieved by changing unsafe $\mathbf{y}$ to its closest safe $\hat{\mathbf{y}}$ by adding a vector $\Delta\mathbf{y}$, which is $\hat{\mathbf{y}} = \mathbf{y} + \Delta\mathbf{y}$. The vector $\Delta\mathbf{y}$ represents the normal vector with the minimum length, among the normal vectors $\alpha^\top$s of the boundary hyperplanes $\alpha^\top\mathbf{x} + \beta = 0$ of the unsafe domain. In the safety properties, we have a common unsafe domain $\mathcal{U}$ bounded by 4 linear constraints and their boundary hyperplanes are respectively, $y_1 - y_2 = 0$, $y_1 - y_3 = 0$, $y_1 - y_4 = 0$ and $y_1 - y_5 = 0$. For a $\mathbf{y} \in \mathcal{U}$, its $\Delta\mathbf{y}$ is computed over these hyperplanes.

The safety of neural networks is first verified with our reachability analysis method. They are determined *safe* if there is no unsafe input space computed on all three safety properties. Then, for unsafe networks that violate at least one of the properties, we conduct the repair process with our framework. The repair process monitors all the properties in case that the model candidate turns unsafe on new properties. 11 of the 45 networks are verified unsafe. The parameter setting for the training of the model in the framework is the same as the ones for the training of the original DNNs. The performance threshold is set to 93%. The experimental results are shown in Table 1. We can see that all the unsafe networks are successfully repaired by our framework, and that compared to the original model, the accuracy changes on the repaired safe model are negligible. There is no obvious performance degradation on the repaired models. This may be because the $\Delta\mathbf{y}$s for the correction is trivial in these cases, seldom impacting accuracy. We can also notice that the network is repaired

efficiently in 3 epochs within totally one minute in most of the cases. It is noteworthy that each repair of the violation on one property does not induce violations on other properties. This is likely due to the fact that the input regions defined in the safety properties are apart from each other and repair of unsafe behaviors over one region hardly affects the correct behaviors over other regions.

Table 1. Repair of neural network controllers for HorizontalCAS. There are 11 unsafe neural networks. **accuracy changes** represents the accuracy difference (%) between the repaired safe model and the original unsafe model. **epochs of repair** represents the number of repair iterations. **running time** represents the computational time for the repair.

Neural Networks	N_{11}	N_{12}	N_{15}	N_{16}	N_{17}	N_{19}	N_{26}	N_{41}	N_{52}	N_{55}	N_{59}
Accuracy Changes(%)	+0.55	+0.75	+1.3	+0.39	+1.54	+0.85	+2.25	+0.45	-0.31	+0.74	+0.39
Epochs of repair	3	3	2	3	14	3	11	2	2	2	2
Runing Time(sec)	24.3	23.1	15.2	31.5	1086.4	59.3	504.2	10.2	11.1	20.7	7.9

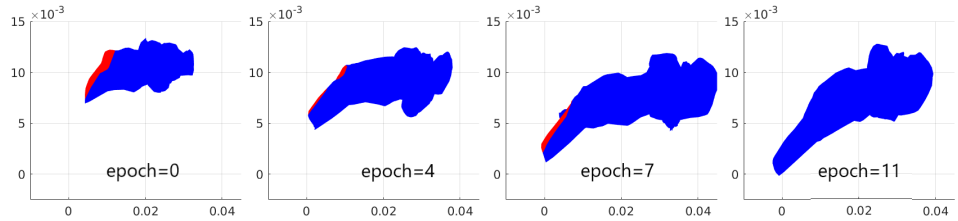


Fig. 8. The evolution of the output reachable domain and the unsafe reachable domain in the repair of DNN N_{26} on property 1. The domains are projected on the output y_1 and y_2 which are respectively, the x axis and y axis. The blue area represents the exact output reachable domain while the red area represents the unsafe reachable domain which is a subset of the exact output reachable domain.

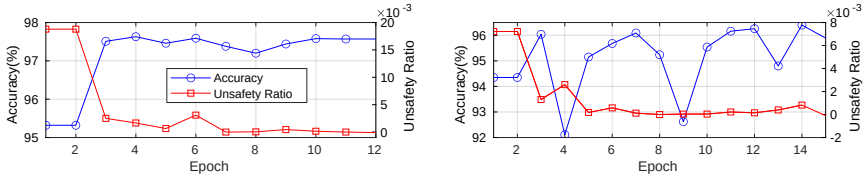


Fig. 9. The evolution of the accuracy and the unsafe input domain in the repair of neural networks N_{26} and N_{17}. The right y axis represents the accuracy of the model candidate. The accuracy refers to the percentage of the correct classification of action advisory. The left y axis represents the approximated volume ratio of the unsafe input domain to the whole input domain specified in the safety property.

More details of the repair process of N_{26} and N_{17} is included to illustrate the process. One is the evolution of the output reachable domain of the candidate model on the violated properties, as shown in Figure 8. We can notice that the reachable domain in blue expands with the repair while the unsafe domain in red gradually disappears. The expansion is mainly because the $\Delta\mathbf{y}$ added to the unsafe $\mathbf{y}$ changes the output distribution of the model candidate by turning away $\mathbf{y}$ from the unsafe domain. The other one is the evolution of the accuracy and the unsafe input spaces of the candidate model, as shown in Figure 9. The unsafe input space computed for each property is quantified by the percentage of the unsafe volume related to the volume of the whole input domain.

Table 2. Comparison of our new reachability analysis method with the method [Yang et al. 2021a] on computational efficiency and memory efficiency. $\mathbf{T}_r(sec)$ and $\mathbf{M}_r(GB)$ denote the computational time and the maximum memory usage of our method for the reachability analysis in one repair. $\mathbf{T}_{nr}(sec)$ and $\mathbf{M}_{nr}(GB)$ are for [Yang et al. 2021a] on the same model candidate models.

Nets	N_{11}	N_{12}	N_{15}	N_{16}	N_{17}	N_{19}	N_{26}	N_{41}	N_{52}	N_{55}	N_{59}
$\mathbf{T}_r(sec)$	3.3	5.5	4.6	5.2	61.6	9.0	24.0	3.1	3.1	5.0	3.3
$\mathbf{T}_{nr}(sec)$	3.5	4.9	5.7	5.1	56.8	9.0	29.3	3.5	3.4	4.6	3.4
$\mathbf{M}_r(GB)$	4.90	4.91	4.97	4.92	4.97	4.97	4.96	4.94	4.94	4.97	4.94
$\mathbf{M}_{nr}(GB)$	4.94	4.97	5.15	4.99	5.23	5.19	5.12	4.99	5.00	5.00	5.01

It can be approximated through a large amount of samplings. We can notice that the tendency of the ratio decreases along with the repair, indicating the unsafe input domain gradually disappears. While the new accuracy seldom goes below the original accuracy, indicating the repair in this cases does not degrade the performance.

Table 2 describes the comparison of our new reachability analysis method to the method [Yang et al. 2021a] regarding computational and memory efficiency. We can notice that there are no obvious difference between the performance of these methods. This is because the computation of the reachability analysis is negligible so that the overhead can not be distinguished.

6.2 Rocket Lander Benchmark

The rocket lander benchmark is based on the lunar lander presented in [Brockman et al. 2016]. It is a vertical rocket landing model simulating SpaceX's Falcon 9 first stage rocket. Unlike the lunar lander whose action space is discrete, the 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 benchmark is shown in Figure 10. 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 un-actuated 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.

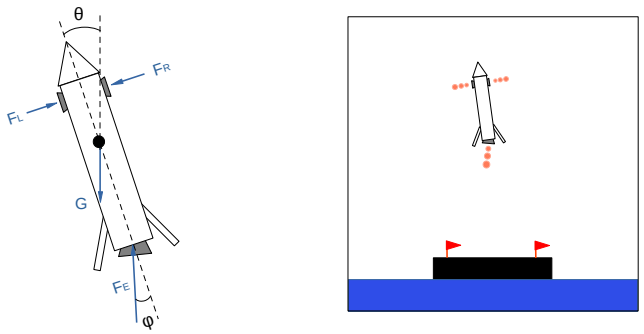


Fig. 10. Rocket lander benchmark.

There are three actions, the main engine thruster F_E, its angle φ and the side nitrogen thrusters F_S. The original observation contains the position x and y of the rocket relative to the landing center on the barge, the velocity v_x and v_y of the rocket, its lateral angle θ, its angular velocity ω. To improve the performance of agents, we also incorporate the last action advisory into the observation for reference. Then, the new observation can denoted as $[x, y, v_x, v_y, \theta, \omega, F'_E, \varphi', F'_S]$. Their lower bound lb and upper bound ub are in Equation 20. The starting state and the reward are similar to the lunar lander The termination conditions of one *episode* includes (1) $|x| > 1$ which indicates the rocket moves out of the barge in x-space, (2) $y > 1.3$ or $y < 0$ which indicates the rocket moves out of the y-space or below the barge, (3) $\theta > 35°$ which indicates the rocket tilts greater than the controllable limit.

$$\begin{aligned} lb &= [-\infty, 0, -\infty, -\infty, -\pi, -\infty, 0, -15°, -1] \\ ub &= [+\infty, +\infty, +\infty, +\infty, \pi, +\infty, 1, 15°, 1] \end{aligned} \tag{20}$$

Two safety properties are defined for the agent as below. Since reachability analysis processes bounded sets, the infinite lower bounds and upper bounds of states above will be replaced with the searched state space in the learning process of the original agent.

(1) *property 1*: for the state constraints $-20° \leq \theta \leq -6°$, $\omega < 0$, $\varphi' \leq 0°$ and $F'_S \leq 0$, the desired action should be $\varphi < 0$ or $F_S < 0$, namely, the unsafe action domain is $\varphi \geq 0 \cap F_S \geq 0$. It describes a scenario where the agent should always stop the rocket from tilting to the right.
(2) *property 2*: for the state constraints $6° \leq \theta \leq 20°$, $\omega \geq 0$, $\varphi' \geq 0°$ and $F'_S \geq 0$, the desired action should be $\varphi > 0$ or $F_S > 0$, namely, the unsafe action domain is $\varphi \leq 0 \cap F_S \leq 0$. It describes a scenario where the agent should always stop the rocket from tilting to the left.

The reinforcement learning algorithm Deep Deterministic Policy Gradients (DDPG) [Lillicrap et al. 2015] 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 rather than probability distribution over them, and Critic, a Q-value network that takes state and action as input and outputs Q-values. The Actor is our target agent controller with its safety properties. DDPG uses experience replay to update Actor and Critic, where in the training process, a set of *tuple*s are sampled from previous experiences.

Here, we first learn several agents with the DDPG algorithm. Then, we apply our framework to repair agents that violate the safety properties. The architecture of the Actor is designed with 9 inputs for state, 5 hidden layers with each containing 20 ReLU neurons, 3 outputs with subsequent tanh function which maps input spaces into $[-1, 1]$. Let the three outputs before the tanh be denoted as y_1, y_2 and y_3, the outputs of Actor are computed by $F_E = 0.5 \times \tanh(y_1) + 0.5$, $\varphi = 15° \times \tanh(y_2)$ and $F_S = \tanh(y_3)$. Our reachability analysis is applied to the architecture before the tanh function, which contains only ReLU neurons. Although the unsafe output domains defined in safety properties above are for outputs φ and F_S after the tanh function, the domains $\varphi \geq 0 \cap F_S \geq 0$, $\varphi \leq 0 \cap F_S \leq 0$ are actually equivalent to $y_2 \geq 0 \cap y_3 \geq 0$, $y_2 \leq 0 \cap y_3 \leq 0$. The architecture of the Critic is designed with 12 inputs for state and action, 5 hidden layers with each containing 20 ReLU neurons, 1 output for the Q-value. The capacity of the global buffer to store previous experience is set to 4×10^5. The learning rate for Actor and Critic is set to 10^{-4} and 10^{-3} respectively. The 1000 *episode*s are performed for each learning. Overall, three unsafe agents are obtained.

For the repair process, the parameters are as follows. the learning rates for Actor and Critic stay unchanged. A buffer stores all old experiences from the learning process. In addition, another global buffer is included to store new experiences with unsafe states as initial states. From these two buffers, old experiences as well as new experiences are randomly selected to form a set of

training *tuple*s in Figure 4(b). As introduced, a new penalty reward is added for any wrong actions generated from input states. Its value is normally set to the lowest reward in the old experience. Here, the penalty is set to -30. To maintain the performance, the threshold of the change ratio which is defined in Equation 21 is set to -0.2.

$$Ratio = \frac{Performance_{(repaired\ agent)} - Performance_{(original\ agent)}}{Performance_{(original\ agent)}} \tag{21}$$

For each unsafe agent, we conduct the repair 5 times with each repair aiming to obtain a safe agent. There are 15 instances used for evaluation. The experimental results are shown in Table 3 and 4. The evolution of the unsafe input domain like Figure 9 is not included for this benchmark because the sampling does not work well for high dimensional input space. Table 3 describes the performance change ratio, the epochs of repair and the total time, where the performance of agents is evaluated by the averaged reward on 1000 episodes of running. 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 epochs for all instances with the running time ranging from 332.7 seconds to 2632.9 seconds. Also, during the repair process, we notice that repairing unsafe behaviors on one property occasionally leads to new unsafe behaviors on the other property. This is likely because the input regions defined in the properties are adjacent to each other but their desired output regions are different, and the repaired behaviors over one input region can easily expand to other regions over which different behaviors are expected.

Next, we conduct a comparison of our new reachability analysis method with the method presented in [Yang et al. 2021a]. The results are shown in Table 4. In terms of computational efficiency and memory efficiency, our method outperforms this method in [Yang et al. 2021a]. The computational efficiency improvement of our method ranges between a maximum of 6.8 times faster and a minimum of 3.6 times faster, with an average of 4.7. The reduction on memory usage ranges between 61.7% and 70.2%, with an average of f 64.5%. It is noteworthy that there are many factors that affect computational complexity of the reachability analysis, such as the number of neurons, the input domain as well as the parameter weights of DNNs. Therefore, for the three agents with the same architecture but different weights, the computational time is different.

Table 3. Repair of unsafe agents for the rocket lander. **ID** is the index of each repair. **Ratio** denotes the performance change ratio of the repaired agent compared to the original unsafe agent as formulated in Equation 21. **Epoch** denotes the number of epochs for repair. **Time** (*sec*) denotes the running time for one repair with our reachability analysis method.

ID	Agent 1			Agent 2			Agent 3		
	Ratio	**Epoch**	**Time**	**Ratio**	**Epoch**	**Time**	**Ratio**	**Epoch**	**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

In addition, we analyze the evolution of the reachable domain of candidate models in the repair. An example of Agent 1 on the first repair is shown in Figure 11. At $epoch = 0$ which is before the repair, we can notice that the candidate agent has unsafe output reachable domain on Property 1 and is safe on Property 2. At $epoch = 1$, the unsafe reachable domain becomes smaller, which indicates that the agent learned from the penalty assigned to the unsafe actions and its action space

Table 4. Comparison of our new reachability analysis method with the method [Yang et al. 2021a] on computational efficiency and memory efficiency. $\mathbf{T}_r(sec)$ and $\mathbf{M}_r(GB)$ denote the computational time and the maximum memory usage of our method for all reachability analysis of all candidate models in one repair. $\mathbf{T}_{nr}(sec)$ and $\mathbf{M}_{nr}(GB)$ are for [Yang et al. 2021a] on the same model candidate models.

	Agent 1				Agent 2				Agent 3			
ID	$\mathbf{T}_r$	$\mathbf{T}_{nr}$	$\mathbf{M}_r$	$\mathbf{M}_{nr}$	$\mathbf{T}_r$	$\mathbf{T}_{nr}$	$\mathbf{M}_r$	$\mathbf{M}_{nr}$	$\mathbf{T}_r$	$\mathbf{T}_{nr}$	$\mathbf{M}_r$	$\mathbf{M}_{nr}$
1	129.4	760.0	15.94	42.69	426.1	1583.6	13.38	36.39	700.5	2513.7	14.48	46.05
2	122.9	740.4	15.48	40.45	935.9	3352.8	13.58	37.23	618.6	2586.6	14.33	48.18
3	206.7	1361.9	16.74	45.41	572.6	2106.7	13.55	36.55	645.3	3054.0	15.13	44.91
4	329.0	2250.6	15.66	43.89	428.4	1569.7	13.67	35.97	714.4	3019.8	15.49	47.10
5	224.53	1454.2	15.32	40.77	579.5	2108.3	13.68	35.99	997.0	3277.3	15.99	48.86

moves towards the safe domain. After another repair, the agent becomes safe on both properties. We can also notice that during the repair the output reachable domains do not change much, indicating that the performance of the agent is preserved.

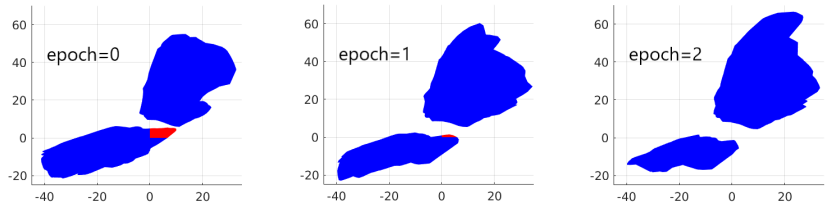


Fig. 11. The evolution of the output reachable domain and the unsafe reachable domain in the repair of Agent 1 on the first repair. x axis represents y_2 and y axis represents y_3. The blue area represents the exact output reachable domain while the red area represents the unsafe reachable domain which is a subset of the exact output reachable domain. The bottom left area is the reachable domain on Property 1 and the top right area is the reachable domain on Property 2.

7 RELATED WORKS

Adversarial and robust training. The adversarial training [Goodfellow et al. 2014; Madry et al. 2017; Mirman et al. 2018; Wong and Kolter 2018; Zhang et al. 2019] is a type of method where adversarial examples are obtained by adversarial attacks or reachability analysis based on the over approximation. These methods have been shown effective in improving the robustness of DNN against adversarial attacks. It inspires us to combine the adversarial training with the more accurate reachability analysis methods that provide complete details of misbehaviors, such that provably safe DNNs can be learned. The difference between our framework and the adversarial training is that instead of examples from random attacks, our framework can identify examples representative of the entire unsafe domain computed from the reachability analysis.

Verification of Deep Neural Networks. Many methods for safety verification of DNNs have been developed, which are mainly based on *reachability* [Gehr et al. 2018; Tran et al. 2019b; Xiang et al. 2017, 2018; Yang et al. 2020], *optimization* [Bastani et al. 2016; Dvijotham et al. 2018; Lomuscio and Maganti 2017; Raghunathan et al. 2018; Tjeng et al. 2019; Wong and Kolter 2018], and *search* [Bunel et al. 2018; Dutta et al. 2018; Ehlers 2017; Huang et al. 2017; Katz et al. 2017; Wang et al. 2018; Weng et al. 2018]. The reachability analysis method is a very appealing method for the repair because it can provide regions of DNN misbehaviors. The spectrum of reachability methods can be broadly categorized in two classes: over-approximation and exact analysis methods.

Over-approximation methods are able to provide safety guarantees but are also *incomplete*, i.e., the method may return *unsafe* due to over-approximation when in fact the system is *safe*. These methods ensure quick safety verification but are not sufficient for the repair. The repair also requires the exact analysis method which can compute the exact unsafe input domain and output domain of DNNs. Thus, we design an algorithm to integrate these two type of method by taking advantage of their merits, which has been shown around 5 times faster than the related work.

The efficiency and accuracy of reachability analysis is strongly associated with the set representation. Approaches, particularly for the DNN's, *Zonotope* [Gehr et al. 2018], *Star-set* [Bak et al. 2020; Tran et al. 2020a, 2019a,c, 2021, 2020b] and *facet-vertex incidence matrix* (FVIM) with vertices [Yang et al. 2021a] are utilized. Each set representation has its advantages and challenges. For instance, a *zonotope* can be represented with finite vectors $\mathbf{v}_i$ by summing $a_i\mathbf{v}_i$, where a_i is scalar ranging between 0 and 1. Such a simple representation enables the development of fast over-approximation methods for reachability analysis. The *star-set* representation is essentially an enhancement of the *half-space representation*, which can efficiently process affine mapping in DNNs. The FVIM is used for exact reachability analysis for repair in this work. For the integration of methods, the new set representation $\mathcal{V}$-zono which can be compatible with FVIM and efficiently encodes vertices is designed for the over approximation method with ReLU linear relaxation.

Deep Neural Networks Repair. Works [Goldberger et al. 2020; Sohn et al. 2019a] attempt to correct unsafe behavior of DNNs by modifying neural weights that is likely associated with the misbehaviors. Due to the black-box nature of DNNs, the modification of such weights may result in unpredicted performance degradation of DNNs. Work [Sotoudeh and Thakur 2021a] introduces a *Decoupled* DNN architecture. Based on this architecture, their provable polytope repair which aims to correct misbehaviors of ReLU DNNs over a domain can be reduced to a LP problem. However, this method is only applicable to the two-dimensional input region for the DNNs having similar size to the ones in the HorizontalCAS benchmark – a five-dimensional input.

8 CONCLUSION AND FUTURE WORK

We have presented a reachability-based framework to repair unsafe DNN controllers for autonomous systems. The approach can be utilized to repair unsafe DNNs with only training data available. It can also be integrated into existing reinforcement algorithms to synthesize safe DNN controllers. Our experimental results on two practical benchmarks have shown that the proposed framework can successfully obtain a provably safe DNN while maintaining its accuracy and performance. We utilize a new set representation and integrate an over approximation method to improve the performance and memory footprint of our rechability analysis algorithm.

Nonetheless, safe training or repairing of DNNs with reachability analysis is still a challenging problem. There are several aspects we plan to study in the future. Firstly, computation of the unsafe set domain of larger-scale DNNs with higher dimensional inputs, such as DNNs for image classification, is still challenging. Therefore, new approaches are needed to repair such unsafe DNNs. Secondly, training of DNNs relies on appropriate meta parameters and cannot always guarantee the convergence to optimal performance, which can impose difficulties for the repair process to converge. Thus, analysis of the interaction between DNN training and its repair is necessary. Furthermore, from our observations it becomes more difficult to repair unsafe DNNs when the input spaces of two safety properties are adjacent but with different desired output behaviors. This is due to the fact that it may be difficult for the DNN to learn a boundary to distinguish these adjacent input spaces and behave correctly. Therefore, a thorough study on understanding the convergence of the proposed framework is critical for enhancing its applicability to real-world applications.

REFERENCES

Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2018. Safe reinforcement learning via shielding. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Greg Anderson, Abhinav Verma, Isil Dillig, and Swarat Chaudhuri. 2020. Neurosymbolic reinforcement learning with formally verified exploration. *arXiv preprint arXiv:2009.12612* (2020).

Anish Athalye, Nicholas Carlini, and David Wagner. 2018. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning*. PMLR, 274–283.

Stanley Bak, Hoang-Dung Tran, Kerianne Hobbs, and Taylor T. Johnson. 2020. Improved Geometric Path Enumeration for Verifying ReLU Neural Networks. In *Proceedings of the 32nd International Conference on Computer Aided Verification*. Springer.

Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. 2016. Measuring neural net robustness with constraints. In *Advances in neural information processing systems*. 2613–2621.

Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. 2020. Efficient Verification of ReLU-Based Neural Networks via Dependency Analysis.. In *AAAI*. 3291–3299.

Maxime Bouton, Jesper Karlsson, Alireza Nakhaei, Kikuo Fujimura, Mykel J Kochenderfer, and Jana Tumova. 2019. Reinforcement learning with probabilistic guarantees for autonomous driving. *arXiv preprint arXiv:1904.07189* (2019).

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. 2018. A unified view of piecewise linear neural network verification. In *Advances in Neural Information Processing Systems*. 4790–4799.

Richard Cheng, Gábor Orosz, Richard M Murray, and Joel W Burdick. 2019. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 3387–3395.

Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. 2018. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*. Springer, 121–138.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A Mann, and Pushmeet Kohli. 2018. A Dual Approach to Scalable Verification of Deep Networks.. In *UAI*, Vol. 1. 2.

Ruediger Ehlers. 2017. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 269–286.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. 2020. Pruning neural networks at initialization: Why are we missing the mark? *arXiv preprint arXiv:2009.08576* (2020).

Nathan Fulton and André Platzer. 2018. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3–18.

Ben Goldberger, Guy Katz, Yossi Adi, and Joseph Keshet. 2020. Minimal Modifications of Deep Neural Networks using Verification.. In *LPAR*, Vol. 2020. 23rd.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

Martin Henk, Jürgen Richter-Gebert, and Günter M Ziegler. 2004. 16 basic properties of convex polytopes. *Handbook of discrete and computational geometry* (2004), 255–382.

Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. 2020. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review* 37 (2020), 100270.

Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. 2017. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 3–29.

Md Johirul Islam, Rangeet Pan, Giang Nguyen, and Hridesh Rajan. 2020. Repairing deep neural networks: Fix patterns and challenges. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1135–1146.

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

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

Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. 2019. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 443–452.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

Alessio Lomuscio and Lalit Maganti. 2017. An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351* (2017).

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* (2017).

Matthew Mirman, Timon Gehr, and Martin Vechev. 2018. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*. 3578–3586.

Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. 2018. Certified Defenses against Adversarial Examples. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Bys4ob-Rb

Sanjit A Seshia, Ankush Desai, Tommaso Dreossi, Daniel J Fremont, Shromona Ghosh, Edward Kim, Sumukh Shivakumar, Marcell Vazquez-Chanlatte, and Xiangyu Yue. 2018. Formal specification for deep neural networks. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 20–34.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 41.

Jeongju Sohn, Sungmin Kang, and Shin Yoo. 2019a. Search based repair of deep neural networks. *arXiv preprint arXiv:1912.12463* (2019).

Jeongju Sohn, Sungmin Kang, and Shin Yoo. 2019b. Search based repair of deep neural networks. *arXiv preprint arXiv:1912.12463* (2019).

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

Matthew Sotoudeh and Aditya V Thakur. 2021b. SyReNN: A Tool for Analyzing Deep Neural Networks. *Tools and Algorithms for the Construction and Analysis of Systems* 12652 (2021), 281.

Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HyGIdiRqtm

Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. 2020. On Adaptive Attacks to Adversarial Example Defenses. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 1633–1645. https://proceedings.neurips.cc/paper/2020/file/11f38f8ecd71867b42433548d1078e38-Paper.pdf

Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. 2020a. Verification of Deep Convolutional Neural Networks Using ImageStars. In *32nd International Conference on Computer-Aided Verification (CAV)*. Springer. https://doi.org/10.1007/978-3-030-53288-8_2

Hoang-Dung Tran, Feiyang Cei, Diego Manzanas Lopez, Taylor T. Johnson, and Xenofon Koutsoukos. 2019a. Safety Verification of Cyber-Physical Systems with Reinforcement Learning Control. In *ACM SIGBED International Conference on Embedded Software (EMSOFT'19)*. ACM.

Hoang-Dung Tran, Diago Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T Johnson. 2019b. Star-Based Reachability Analysis of Deep Neural Networks. In *International Symposium on Formal Methods*. Springer, 670–686.

Hoang-Dung Tran, Patrick Musau, Diego Manzanas Lopez, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T. Johnson. 2019c. Star-Based Reachability Analsysis for Deep Neural Networks. In *23rd International Symposisum on Formal Methods (FM'19)*. Springer International Publishing.

Hoang-Dung Tran, Neelanjana Pal, Patrick Musau, Xiaodong Yang, Nathaniel P. Hamilton, Diego Manzanas Lopez, Stanley Bak, and Taylor T. Johnson. 2021. Robustness Verification of Semantic Segmentation Neural Networks using Relaxed Reachability. In *33rd International Conference on Computer-Aided Verification (CAV)*. Springer.

Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T Johnson. 2020b. NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification*. Springer, 3–17.

Caterina Urban, Maria Christakis, Valentin Wüstholz, and Fuyuan Zhang. 2020. Perfectly Parallel Fairness Certification of Neural Networks. *Proceedings of the ACM on Programming Languages* 4, OOPSLA (2020), 185:1–185:30. https://doi.org/10.1145/3428253

Caterina Urban and Antoine Miné. 2021. A Review of Formal Methods applied to Machine Learning. *arXiv preprint arXiv:2104.02466* (2021).

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal security analysis of neural networks using symbolic intervals. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1599–1614.

Zi Wang, Aws Albarghouthi, and Somesh Jha. 2020. Abstract Universal Approximation for Neural Networks. *arXiv e-prints* (2020), arXiv–2007.

Lily Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. 2018. Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learning*.

PMLR, 5276–5285.

Eric Wong and Zico Kolter. 2018. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*. PMLR, 5286–5295.

Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. 2017. Reachable set computation and safety verification for neural networks with relu activations. *arXiv preprint arXiv:1712.08163* (2017).

Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. 2018. Output reachable set estimation and verification for multilayer neural networks. *IEEE transactions on neural networks and learning systems* 29, 11 (2018), 5777–5783.

Zikang Xiong, Joe Eappen, He Zhu, and Suresh Jagannathan. 2020. Robustness to Adversarial Attacks in Learning-Enabled Controllers. *arXiv preprint arXiv:2006.06861* (2020).

Zikang Xiong and Suresh Jagannathan. 2021. Scalable Synthesis of Verified Controllers in Deep Reinforcement Learning. *arXiv preprint arXiv:2104.10219* (2021).

Xiaodong Yang, Taylor T Johnson, Hoang-Dung Tran, Tomoya Yamaguchi, Bardh Hoxha, and Danil Prokhorov. 2021a. Reachability Analysis of Deep ReLU Neural Networks Using Facet-Vertex Incidence. In *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control* (Nashville, Tennessee) *(HSCC '21)*. Association for Computing Machinery, New York, NY, USA, Article 18, 7 pages. https://doi.org/10.1145/3447928.3456650

Xiaodong Yang, Hoang-Dung Tran, Weiming Xiang, and Taylor Johnson. 2020. Reachability Analysis for Feed-Forward Neural Networks using Face Lattices. *arXiv preprint arXiv:2003.01226* (2020).

Xiaodong Yang, Tomoya Yamaguchi, Hoang-Dung Tran, Bardh Hoxha, Taylor T Johnson, and Danil Prokhorov. 2021b. Reachability Analysis of Convolutional Neural Networks. *arXiv preprint arXiv:2106.12074* (2021).

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. 2018. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems*. 4944–4953.

Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. 2019. Theoretically principled trade-off between robustness and accuracy. In *International Conference on Machine Learning*. PMLR, 7472–7482.

Weichao Zhou, Ruihan Gao, BaekGyu Kim, Eunsuk Kang, and Wenchao Li. 2020. Runtime-safety-guided policy repair. In *International Conference on Runtime Verification*. Springer, 131–150.