Input-Relational Verification of Deep Neural Networks

DEBANGSHU BANERJEE, University of Illinois at Urbana-Champaign, USA
CHANGMING XU, University of Illinois at Urbana-Champaign, USA
GAGANDEEP SINGH, University of Illinois at Urbana-Champaign, USA and VMware Research, USA

We consider the verification of input-relational properties defined over deep neural networks (DNNs) such as robustness against universal adversarial perturbations, monotonicity, etc. Precise verification of these properties requires reasoning about multiple executions of the same DNN. We introduce a novel concept of difference tracking to compute the difference between the outputs of two executions of the same DNN at all layers. We design a new abstract domain, DiffPoly for efficient difference tracking that can scale large DNNs. DiffPoly is equipped with custom abstract transformers for common activation functions (ReLU, Tanh, Sigmoid, etc.) and affine layers and can create precise linear cross-execution constraints. We implement an input-relational verifier for DNNs called RaVeN which uses DiffPoly and linear program formulations to handle a wide range of input-relational properties. Our experimental results on challenging benchmarks show that by leveraging precise linear constraints defined over multiple executions of the DNN, RaVeN gains substantial precision over baselines on a wide range of datasets, networks, and input-relational properties.

CCS Concepts: • **Theory of computation** → **Program verification**; **Abstraction**; • **Computing methodologies** → **Neural networks**.

Additional Key Words and Phrases: Abstract Interpretation, Deep Learning, Relational Verification

ACM Reference Format:
Debangshu Banerjee, Changming Xu, and Gagandeep Singh. 2024. Input-Relational Verification of Deep Neural Networks. *Proc. ACM Program. Lang.* 8, PLDI, Article 147 (June 2024), 27 pages. https://doi.org/10.1145/3656377

1 INTRODUCTION

Deep neural networks (DNNs) have become more powerful and widespread over the past few years and have now penetrated almost all fields and application areas including safety-critical domains such as autonomous driving [10] or medical diagnosis [2], etc. Especially in these domains, the decisions generated from these DNNs are important and mistakes can have grave consequences. However, it can be hard to reason about DNNs as they are constructed in a black-box manner and have highly nonlinear behavior. As such, although the machine learning community has made great strides towards discovering and defending against DNN vulnerabilities [33, 50, 54, 60, 72, 84], these methods cannot guarantee safety. As a result, there has been a lot of work on verifying the safety properties of DNNs [3, 4, 6, 13, 14, 22, 32, 38, 39, 43, 56–58, 67, 68, 70, 75, 76, 82, 83, 86, 87, 89]. Despite this progress, existing DNN verification techniques can be imprecise for input-relational properties that arise in many practical scenarios. For example, most existing works mentioned above focus on verifying the absence of an adversarial attack (imperceptible perturbations added to an input) around a local neighborhood of test inputs. Recent work [46] has shown that attacks against individual inputs can be unrealistic as they rely on the attacker having perfect knowledge

Authors' Contact Information: Debangshu Banerjee, University of Illinois at Urbana-Champaign, Urbana, USA, db21@illinois.edu; Changming Xu, University of Illinois at Urbana-Champaign, Urbana, USA, cx23@illinois.edu; Gagandeep Singh, University of Illinois at Urbana-Champaign, Urbana, USA and VMware Research, Palo Alto, USA, ggnds@illinois.edu.

ACM 2475-1421/2024/6-ART147
https://doi.org/10.1145/3656377

of the inputs processed by the DNN and being able to create perturbations specialized for that input. Indeed, many practical attack scenarios [46, 47, 49] involve constructing universal adversarial perturbations (UAPs) [54] that can work against a set of inputs. Other interesting input-relational properties that have become popular in recent years include monotonicity [74], and fairness [40]. Efficient verification of input-relational properties requires reasoning about the relationship between multiple executions of the same DNN. Existing verifiers lack these capabilities and as a result, are not precise. For the remainder of this paper, relational will refer to input-relational.

This Work. In this work, we propose a framework for verifying the relational properties of DNNs - RaVeN (Relational Verifier of Neural Networks). To the best of our knowledge, RaVeN is the first framework to verify a broad range of relational properties defined over multiple executions of the same DNN. Next, we detail the key technical contributions that allow RaVeN to verify relational properties that state-of-the-art verifiers [68, 69, 88] cannot.

Main Contributions. Our main contributions are:

- A new abstract domain, DiffPoly with custom abstract transforms for affine and activation (ReLU, Sigmoid, Tanh, etc.) layers allowing us to efficiently compute precise lower and upper bounds of the difference between the outputs of a pair of DNN executions at each layer.
- A verification framework, RaVeN, which leverages the DiffPoly analysis to compute precise layerwise linear constraints over outputs from different executions of the DNN. These cross-execution linear constraints allow us to capture linear dependencies between the outputs of different DNN executions at each layer, making RaVeN more precise than existing state-of-the-art verifiers [68, 69, 88] which do not track linear dependencies at all layers. We use the linear constraints from DiffPoly analysis to formulate a mixed-integer linear program (MILP) (Section 4). We formally prove the soundness of RaVeN in Section 4.7.
- A complete implementation of RaVeN, including DiffPoly and MILP formulations capable of handling diverse relational properties defined over the same DNNs with the popular feedforward architectures and common activation functions like ReLU, Sigmoid, Tanh, etc.
- An extensive evaluation of RaVeN on a range of popular datasets, challenging fully-connected and convolutional networks, and diverse relational properties (e.g., UAP verification, monotonicity). Our results demonstrate that RaVeN achieves notably higher precision compared to prior approaches and can verify relational properties that are beyond the capabilities of current state-of-the-art verifiers (Section 5).

Our research can serve as a foundation for advancing relational verification in DNNs. Notably, our results indicate that DNNs exhibit improved provable robustness against universal attacks (UAPs), which are more realistic, compared to individual attacks. Recent studies [49, 85] demonstrate that defending against UAPs enhances accuracy and empirical robustness more effectively than defending against individual attacks [50]. In the future, integrating RaVeN into the training loop [52, 55, 90] can lead to DNNs with superior accuracy and provable robustness against UAPs. The supplementary materials[1] and code[2] are publicly available.

2 BACKGROUND

In this section, we present the essential background and notation used in this paper. Throughout the subsequent sections, lowercase letters (a, b, etc.) denote scalars, while uppercase letters (A, B, etc.) and the over barred lowercase letters ($\bar{a}, \bar{b}$, etc.) represent vectors and matrices.

Neural Networks: We primarily focus on feed-forward neural networks. However, since we use linear bound propagation techniques, similar to [86], our method can be extended to other

[1] The latest version of the paper with appendix can be found at https://focallab.org/files/raven.pdf

[2] The code for RaVeN can be found at https://github.com/uiuc-focal-lab/RaVeN.

architectures that can be expressed as DAGs (directed acyclic graphs). We use "DNN" to refer specifically to feed-forward neural networks. These DNNs, denoted as $N : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_l}$, are composed of l sequential layers $N_1, \dots, N_l$, where each $N_i : \mathbb{R}^{n_{i-1}} \rightarrow \mathbb{R}^{n_i}$ is a function. Each layer N_i applies either an affine function (convolution or linear function) or a non-linear activation function, such as ReLU, Sigmoid, or Tanh. Affine layers, represented as $N_i : \mathbb{R}^{n_{i-1}} \longrightarrow \mathbb{R}^{n_i}$, are defined by $N_i(x) = A_i \cdot X + B_i$, where A_i is the weight matrix, and B_i is the bias vector.

2.1 Relational Verification of DNN

For a network $N : R^{n_0} \rightarrow R^{n_l}$ and a relational property defined over DNN inferences on k inputs, the input specification $\Phi : \mathbb{R}^{n_0 \times k} \rightarrow \{true, false\}$ is a boolean predicate. It encodes the input region $\Phi_t \subseteq \mathbb{R}^{n_0 \times k}$ encompassing all potential inputs corresponding to each of the k DNN inferences. For any $X \in \mathbb{R}^{n_0 \times k}$ satisfying Φ, $X = (X_1, \dots, X_k)$ is a tuple of k points where $\forall i \in [k].X_i \in R^{n_0}$ and X_i is the input of the i-th DNN inference. Common DNN relational properties e.g. UAP verification [88], monotonicity [74], etc. can be encoded as the conjunction of k individual input specifications $\phi_{in}^i : \mathbb{R}^{n_0} \rightarrow \{true, false\}$ and cross-execution input specification $\Phi^\delta : \mathbb{R}^{n_0 \times k} \rightarrow \{true, false\}$. Each $\phi_{in}^i : \mathbb{R}^{n_0} \rightarrow \{true, false\}$ defines the input region $\phi_t^i \subseteq \mathbb{R}^{n_0}$ for i-th execution. Meanwhile, Φ^δ captures relationships between inputs used in distinct executions. Commonly Φ^δ bounds the difference between any pair of inputs $X_i, X_j \in \mathbb{R}^{n_0}$ used in different executions such as $L_{i,j} \leq X_i - X_j \leq U_{i,j}$ where $L_{i,j}, U_{i,j} \in \mathbb{R}^{n_0}$ are constant real vectors. Individual input regions ϕ_t^i are in general L_∞ regions [16] i.e. all $X_i \in \mathbb{R}^{n_0}$ such that $\|X_i - X_i^*\|_\infty \leq \epsilon$ around a concrete point $X_i^* \in \mathbb{R}^{n_0}$ with $\epsilon \in \mathbb{R}^+$. For any pair of inputs $X_i, X_j \in \mathbb{R}^{n_0}$, the cross-execution input specification between them $\phi_{i,j}^\delta$ are given by - $\phi_{i,j}^\delta(X_i, X_j) = (L_{i,j} \leq X_i - X_j) \wedge (X_i - X_j \leq U_{i,j})$. The output specification for relational properties is a boolean predicate $\Psi : \mathbb{R}^{n_l \times k} \rightarrow \{true, false\}$ defined over the outputs of all k DNN inferences. In this work, we consider output specifications Ψ that can be expressed as a logical formula in CNF (conjunctive normal form) with m clauses where each clause ψ_i is of the form below $C_{i,j,i'} \in \mathbb{R}^{n_l}$:

$$\psi_i(Y_1, \dots, Y_k) = \bigvee_{j=1}^{n} \psi_{i,j}(Y_1, \dots, Y_k) \quad \text{where } \psi_{i,j}(Y_1, \dots, Y_k) = \left(\sum_{i'=1}^{k} C_{i,j,i'}^T Y_{i'} \geq 0 \right)$$

Definition 2.1 (DNN Relational Verification Problem). The **relational verification** problem for a DNN $N : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_l}$, an input specification $\Phi : \mathbb{R}^{n_0 \times k} \rightarrow \{true, false\}$ and an output specification $\Psi : \mathbb{R}^{n_l \times k} \rightarrow \{true, false\}$ is to prove whether $\forall X_1, \dots, X_k \in \mathbb{R}^{n_0}.\Phi(X_1, \dots, X_k) \implies \Psi(N(X_1), \dots N(X_k))$ or provide a counterexample otherwise.

2.2 Interesting Relational Properties of DNNs

UAP Verification. UAP verification problem verifies whether there exists a single perturbation that can be added to k DNN inputs to make it misclassify all of them. The UAP verification problem is fundamentally different from the commonly considered local L_∞ robustness verification where the adversary can perturb each input independently. However, as shown in recent studies [46, 47, 49] generating input-specific adversarial perturbation is unrealistic, and practical attacks require finding adversarial perturbation that works for a set of inputs instead of a single input. These works suggest that considering robustness against input-specific adversarial attacks is too conservative and presents a pessimistic view of practical DNN robustness. Since the adversarial perturbation is common across a set of inputs, the UAP verification problem requires a relational verifier that can exploit the dependency between perturbed inputs. We provide the input specification Φ and the output specification Ψ of the UAP verification problem in Appendix 8.3. We describe another variation of UAP: targeted UAP in Appendix 8.4.

Worst-case UAP Accuracy. In general, for a given N, finding an adversarial perturbation that works for all inputs in a set is hard. However, an adversarial perturbation affecting a significant proportion of inputs also poses a threat to the DNN. Hence, most of the existing works compute the worst-case accuracy [88] of the DNN on an input set in the presence of a UAP adversary. The formal definition of worst-case UAP accuracy is as follows.

Definition 2.2 (Worst-case UAP Accuracy). Given a DNN N, a set of inputs $I = \{X_1, \ldots, X_k\}$, target outputs $O = \{Y_1, \ldots, Y_k\}$ and perturbation norm bound $\epsilon \in \mathbb{R}$ the worst case UAP of N is $a^* = 1/k \min_{\|V\|_\infty \leq \epsilon} \sum_{i=1}^{k} (N(X_i + V) = Y_i)$ where V is the added perturbation.

Monotonicity Verification. Recent works have shown that local monotonicity of DNNs is interesting and verification for monotonic properties is desirable [21, 61]. This property asserts a monotonic relationship between an input feature and the output. For instance, in predicting housing prices, a monotonic property could stipulate that a house with more rooms is consistently more expensive than a house with fewer rooms. We encode monotonicity as a relational property over a pair of DNN executions in Appendix 8.6.

Hamming Distance. The Hamming distance between two strings is the number of substitutions needed to turn one string into the other [36]. Given a binary string (a list of images of binary digits), we want to formally verify the worst-case bounds on the hamming distance between the original binary string and classified binary string where each image of the binary digits can be perturbed by a common perturbation (formal definition in Appendix 8.5). Hamming Distance serves as a valuable metric for tasks involving input string processing [62], like text comprehension or CAPTCHA solving.

Further Relational Verification Problems. Other than the properties described above, another interesting DNN property is fairness verification [40]. In fairness verification, we want to show a change in a sensitive feature does not change the output (i.e. the model is fair and unbiased towards that feature). We can encode the problem similarly to the monotonicity verification problem presented in the paper and verify it using RaVeN.

3 OVERVIEW

Fig. 1 illustrates the high-level idea behind the workings of RaVeN. It takes, as input, the DNN N and a relational property (Φ, Ψ) defined over k inferences of N. RaVeN computes a product DNN with k copies (one for each inference) of network N and runs existing DNN abstract interpreters [68, 69, 87] on each copy of N to obtain concrete lower and upper bounds of each variable in the product DNN. However, the existing abstract interpreters analyze each DNN execution in isolation and as a result, fail to preserve the dependencies between outputs of different DNN executions. One of our key contributions is the design of a new abstract domain DiffPoly that can efficiently compute precise lower and upper bounds on differences between the outputs of a neuron corresponding to two DNN executions. While DiffPoly can be extended to track bounds on any linear combination of the layerwise outputs of any k DNN executions (Appendix 14.5), we specifically focus on a pair of executions and track differences, not alternatives (e.g., sum), between them. This choice is motivated by the fact that for existing DNN relational properties (UAP verification, monotonicity, etc.), the difference between inputs used in multiple executions is bounded. Therefore, we naturally opt to track differences between the DNN's outputs across multiple executions at subsequent hidden layers and the output layer. RaVeN combines the analysis of existing abstract interpreters on the product DNN and DiffPoly analysis on all $\binom{k}{2}$ pair of executions to infer linear constraints over the outputs of all k executions at each layer. The linear constraints computed by RaVeN capture the dependencies between different DNN executions at each layer making RaVeN more precise than the state-of-the-art relational verifier [88] that only tracks dependencies at the input layer but not

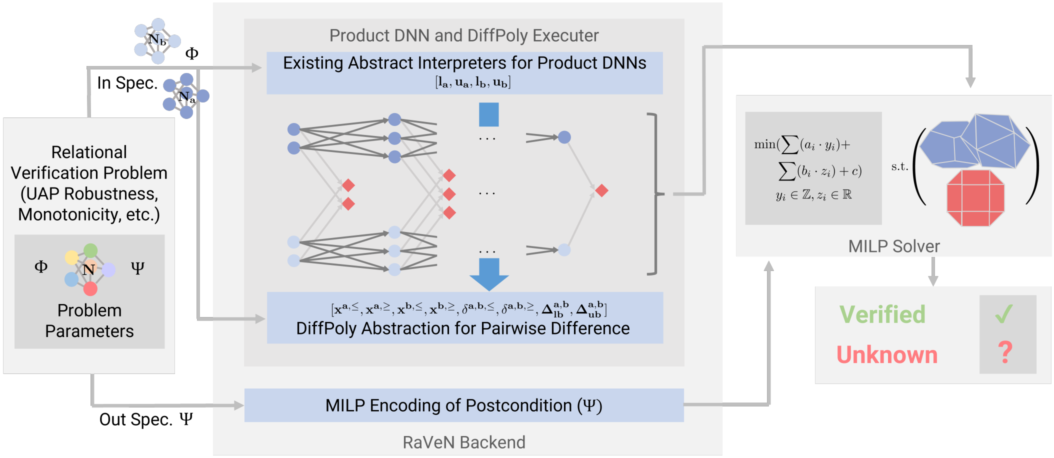

N_b
Φ
In Spec.
N_a
Relational Verification Problem (UAP Robustness, Monotonicity, etc.)
Φ
N
Ψ
Problem Parameters
Out Spec. Ψ
Product DNN and DiffPoly Executer
Existing Abstract Interpreters for Product DNNs
$[\mathbf{l_a}, \mathbf{u_a}, \mathbf{l_b}, \mathbf{u_b}]$
$[\mathbf{x}^{\mathbf{a},\leq}, \mathbf{x}^{\mathbf{a},\geq}, \mathbf{x}^{\mathbf{b},\leq}, \mathbf{x}^{\mathbf{b},\geq}, \delta^{\mathbf{a},\mathbf{b},\leq}, \delta^{\mathbf{a},\mathbf{b},\geq}, \mathbf{\Delta}^{\mathbf{a},\mathbf{b}}_{\mathbf{lb}}, \mathbf{\Delta}^{\mathbf{a},\mathbf{b}}_{\mathbf{ub}}]$
DiffPoly Abstraction for Pairwise Difference
MILP Encoding of Postcondition (Ψ)
RaVeN Backend
$\min(\sum(a_i \cdot y_i)+$
$\sum(b_i \cdot z_i)+c)$
$y_i \in \mathbb{Z}, z_i \in \mathbb{R}$
s.t.
MILP Solver
Verified ✓
Unknown ?

Fig. 1. The overview of the proposed sound and incomplete RaVeN verifier. Given a network N and a relational property (Φ, Ψ) relating k DNN inferences we show the flow of RaVeN along with the key steps - (i) constructing the product DNN by duplicating N k times and analyzing the product DNN with an existing DNN abstract interpreter, (ii) computing pairwise differences of outputs of all k inferences at each layer with DiffPoly analysis that uses concrete lower and upper bounds of each variable in the product DNN, (iii) combining DiffPoly analysis and product DNN analysis with an existing DNN abstract interpreter to infer layerwise linear constraints over outputs of all k DNN executions that preserves dependencies between different DNN executions, (iv) encoding the postcondition as a MILP objective and formulate MILP with layerwise linear constraints computed in step (iii). Finally, we use an off-the-shelf MILP solver [35] to verify the relational property by solving the corresponding MILP.

at the hidden layers and loses precision as a result. At the final layer of N, we encode the output specification Ψ as a set of mixed-integer linear programming (MILP) constraints over the outputs of all k executions. We use integer variables only to encode the output specification Ψ to limit the number of integer variables in the MILP formulation and subsequently avoid exponential blowup in MILP optimization time. Next, we show the workings of RaVeN with an illustrative example.

3.1 Illustrative Example

3.1.1 Network. For this example, we consider the network, N_{ex}, with three layers: two affine layers and one ReLU layer with two neurons each (Fig. 2). The weights on the edges represent the coefficients of the weight matrix used by the affine transformations applied at each layer and the learned bias for each neuron is shown above or below it. N_{ex} can be viewed as a loop-free straight-line program composed of a sequence of assignment statements - ReLU assignments $x_i \leftarrow max(0, x_j)$ and affine assignments $x_i \leftarrow v + \sum_{j=1}^{n} w_j \cdot x_j$ where $v \in \mathbb{R}$ and $W = [w_1, \ldots, w_n]^T \in \mathbb{R}^n$. In the example, N_{ex} is a program with 12 variables: 2 input variables - $\{i_1, i_2\}$, two output variables - $\{o_1, o_2\}$, 8 intermediate variables $\{x_1, \ldots, x_8\}$ and a sequence assignment statements shown below:

$$\begin{array}{lllll} x_1 \leftarrow i_1 & x_3 \leftarrow x_1 - x_2 & x_5 \leftarrow \max(0, x_3) & x_7 \leftarrow x_5 - x_6 & o_1 \leftarrow x_7 \\ x_2 \leftarrow i_2 & x_4 \leftarrow -2 \cdot x_1 + x_2 & x_6 \leftarrow \max(0, x_4) & x_8 \leftarrow -x_5 + x_6 & o_2 \leftarrow x_8 \end{array} \tag{1}$$

3.1.2 Relational Property. We verify the UAP verification problem described in Section 2.2 on N_{ex} where the relational property is defined over 2 separate executions of N_{ex}. Here the input specification $\forall X_1, X_2 \in \mathbb{R}^2 . \Phi(X_1, X_2)$ is defined as follows where $X_1^* = [14, 11]^T$, $X_2^* = [11, 14]^T$, and $\epsilon = 6$.

$$\Phi(X_1, X_2) = (\|X_1 - X_1^*\|_\infty \leq \epsilon) \wedge (\|X_2 - X_2^*\|_\infty \leq \epsilon) \wedge (X_1 - X_2 = X_1^* - X_2^*) \tag{2}$$

In UAP verification, an adversary can select to attack the DNN with any perturbation δ such that $\|\delta\|_\infty \leq \epsilon$ but the same perturbation δ must be applied to both inputs - X_1^*, X_2^*. Therefore the two executions are related and tracking this relationship improves precision. In contrast, in the common local robustness problem, an adversary can choose different perturbations for the two inputs and therefore the two executions are unrelated and can be verified independently. Any input $X_1 \in \mathbb{R}^2$ inside the L_∞ ball defined by $\|X_1 - X_1^*\|_\infty \leq \epsilon$ is not misclassified if $(N_{ex}(X_1) = [o_1, o_2]^T) \wedge (o_1 - o_2 \geq 0)$ holds. Conversely, any input $X_2 \in \mathbb{R}^2$ lying inside the L_∞ ball - $\|X_2 - X_2^*\|_\infty \leq \epsilon$ is not misclassified if $(N_{ex}(X_2) = [o_1, o_2]^T) \wedge (o_2 - o_1 \geq 0)$ holds. We want to formally verify that there does not exist an adversarial perturbation $\delta \in \mathbb{R}^2$ with $\|\delta\|_\infty \leq \epsilon$ such that both the inferences on inputs $X_1 = X_1^* + \delta$ and $X_2 = X_2^* + \delta$ produces incorrect classification results. In this case, the output specification Ψ can be encoded such that $\forall \delta \in \mathbb{R}^2$ and $\|\delta\|_\infty \leq \epsilon$ the network N_{ex} correctly classifies at least one of the two perturbed inputs $X_1 = X_1^* + \delta$ and $X_2 = X_2^* + \delta$.

$$\Psi(N_{ex}(X_1), N_{ex}(X_2)) = (C_1^T N_{ex}(X_1) \geq 0) \vee (C_2^T N_{ex}(X_2) \geq 0) \quad \text{where } C_1 = [1, -1]^T \wedge C_2 = [-1, 1]^T$$

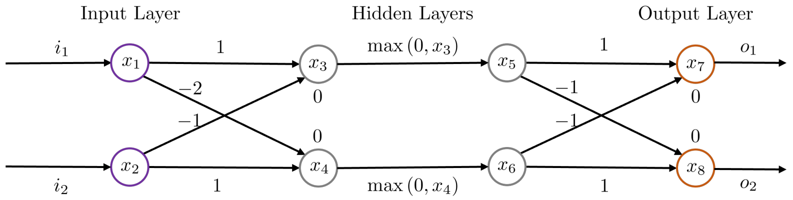


Fig. 2. Representation of N_{ex} used in the illustrative example

3.1.3 Product DNN Construction & Analysis. The input specification Φ (Eq. 2) relates two DNN executions on inputs from two input regions ϕ_t^1, ϕ_t^2 (not necessarily disjoint) defined by $\forall X_1 \in \mathbb{R}^2.\|X_1 - X_1^*\|_\infty \leq \epsilon$ and $\forall X_2 \in \mathbb{R}^2.\|X_2 - X_2^*\|_\infty \leq \epsilon$ respectively. So we construct the product DNN with two separate copies of the DNN - N_{ex}^1 and N_{ex}^2 where N_{ex}^1 and N_{ex}^2 track execution of N_{ex} on inputs from ϕ_t^1 and ϕ_t^2 respectively. The product DNN construction involves maintaining two separate copies of all 12 variables and all 10 assignment statements used in N_{ex}. In the product DNN, for each network N_{ex}^j where $j \in \{1, 2\}$, we rename input variables as $\{i_1^j, i_2^j\}$, output variables as $\{o_1^j, o_2^j\}$ and intermediate variables as $\{x_1^j, \ldots, x_8^j\}$. N_{ex}^1 and N_{ex}^2 can be analyzed with any existing complete [24, 39] or incomplete DNN verifiers [69, 87]. However, for scalability, we use sound but incomplete abstract interpretation-based DNN verification techniques. We use the existing DeepZ [68] abstract interpreter to compute an overapproximated range of the possible values of each variable in N_{ex}^1 and N_{ex}^2 w.r.t. input regions ϕ_t^1 and ϕ_t^2 respectively. Fig. 12 in the appendix shows the range of values for each variable in the product DNN obtained by DeepZ analysis. The detailed execution of DeepZ for this example is in Appendix 8.7.

3.1.4 Capturing Dependencies between DNN Executions. DeepZ (or, any other existing non-relational DNN verifier) analyze N_{ex}^1, N_{ex}^2 in isolation and do not track the relation captured in the cross-execution input constraint such as in Eq. 2 $\forall X_1, X_2.(X_1 - X_2 = X_1^* - X_2^*)$ that bounds the difference between the inputs used in different executions of the network. In contrast, the proposed DiffPoly can efficiently compute the bounds on the difference between two copies of the same variable corresponding to two different executions and as a result, can capture the dependencies between multiple executions. For example, given any variable x_i in N_{ex} DiffPoly computes lower and upper bound of $(x_i^1 - x_i^2)$ that holds for all possible inputs satisfying Φ. Overall, for any relational property defined over k DNN executions, we run $\binom{k}{2}$ DiffPoly for each pair of DNN executions. Note that since for any variable x_i, $(x_i^a - x_i^b) = -(x_i^b - x_i^a)$, for any pair of execution over inputs from ϕ_t^a, and ϕ_t^b, we only run DiffPoly analysis if $a < b$ to avoid redundant computations. For the rest of the paper, given a pair of variables $< x_i^a, x_i^b >$ we use $\delta_{x_i}^{a,b}$ to denote their difference $(x_i^a - x_i^b)$.

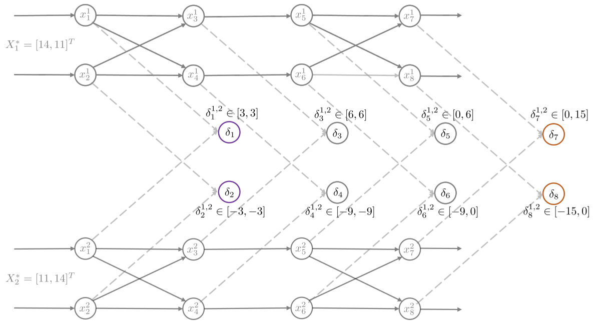
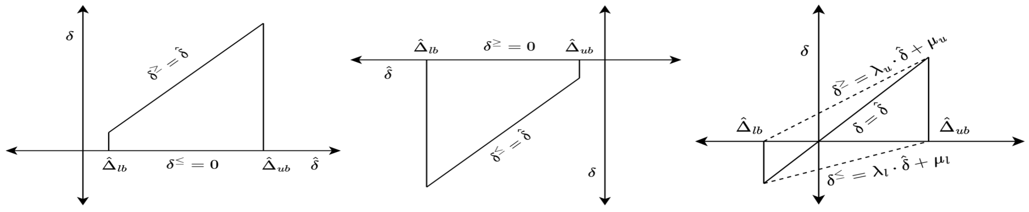


Fig. 3. Concrete bounds of difference as computed by DiffPoly analysis on the example network.

3.1.5 DiffPoly Domain. For two copies of the same variable from two separate executions e.g. x_i^a, x_i^b, the DiffPoly domain (formally described in Section 4.1), associates six linear constraints with $< x_i^a, x_i^b >$: three upper linear constraints (symbolic upper bounds) $\delta_{x_i}^{a,b,\geq}$, $x_i^{a,\geq}$, $x_i^{b,\geq}$ and three lower linear constraints (symbolic lower bounds) $\delta_{x_i}^{a,b,\leq}$, $x_i^{a,\leq}$, $x_i^{b,\leq}$. The δ-constraints are the symbolic lower and upper bound on the difference $(x_i^a - x_i^b)$ satisfying $\delta_{x_i}^{a,b,\leq} \leq (x_i^a - x_i^b) \leq \delta_{x_i}^{a,b,\geq}$ while the other four constraints represent symbolic bounds on the variables x_i^a, x_i^b respectively. Additionally, the domain tracks concrete bounds - concrete lower bounds for each variable $(x_i^a - x_i^b)$, x_i^a, and x_i^b i.e. Δ_{lb}^{a,b,x_i}, l_{a,x_i}, and l_{b,x_i} and concrete upper bounds Δ_{ub}^{a,b,x_i}, u_{a,x_i}, and u_{b,x_i}. Note that as depicted in Fig 1, the concrete bounds - l_{a,x_i}, and l_{b,x_i} u_{a,x_i}, and u_{b,x_i} are obtained from the analysis of the product DNN. At a high level, DiffPoly combines the ideas from the Zone domain [51], used for classical program analysis, that tracks concrete lower and upper bound on the difference of a pair of variables e.g. $l_{xy} \leq (x - y) \leq u_{xy}$ and the DeepPoly domain [69] that tracks symbolic lower and upper bound on variables of the DNN. However, DiffPoly is more precise than both the Zone domain which does not track symbolic bounds on the difference, and the DeepPoly domain which does not explicitly track any difference constraints making DiffPoly well suited for computing difference bounds across multiple DNN executions. Next, we show the format of symbolic bounds associated with DiffPoly below where $\delta_{x_j}^{a,b} = (x_j^a - x_j^b)$.

$$\delta_{x_i}^{a,b,\geq} = v + \sum_{j=1}^{n} \left(w_j^{\delta} \cdot \delta_{x_j}^{a,b} + w_j^a \cdot x_j^a + w_j^b \cdot x_j^b \right) \quad x_i^{a,\geq} = v_a^x + \sum_{j=1}^{n} w_j^{a,x} \cdot x_j^a \quad x_i^{b,\geq} = v_b^x + \sum_{j=1}^{n} w_j^{b,x} \cdot x_j^b \qquad (3)$$

In Eq. 3, $v, v_a^x, v_b^x \in \mathbb{R}$, $W^{\delta}, W^a, W^b, W^{a,x}, W^{b,x} \in \mathbb{R}^n$ are the coefficients of the variables with w_i denoting the i-th coefficient for any vector $W \in \mathbb{R}^n$, n is the number of neurons in N_{ex}. We restrict the format of symbolic bounds and enforce $\forall j \geq i$ $w_j^{\delta} = w_j^a = w_j^b = w_j^{a,x} = w_j^{b,x} = 0$ so that symbolic bounds of any pair of variables $< x_i^a, x_i^b >$ involve only variables that come before x_i^a, x_i^b (having smaller index) and their difference. These restrictions ensure that there are no cyclic dependencies between the symbolic bounds of the variables. Moreover, similar to the DeepPoly



(a) $\hat{\Delta}_{lb} \geq 0$ (b) $\hat{\Delta}_{ub} \leq 0$ (c) $\hat{\Delta}_{lb} < 0 \wedge \hat{\Delta}_{ub} > 0$

Fig. 4. The optimal (in terms of area) convex approximations for $\delta = ReLU(x) - ReLU(y)$ where $\hat{\delta} = (x - y)$, $\delta^{\geq}$, and $\delta^{\leq}$ are symbolic upper bound and lower bound of δ respectively.

domain, we only allow a single symbolic lower, and upper bound to reduce the computation cost required to evaluate the concrete bounds for each variable. Otherwise, the unrestricted Polyhedra domain [20] though more precise, does not scale to the large DNNs considered in this work.

3.1.6 DiffPoly Analysis. The analysis start with computing the symbolic and concrete bounds corresponding to $< x_1^1, x_1^2 >$ and $< x_2^1, x_2^2 >$. All pair of inputs X_1, X_2 satisfying input specification Φ satisfy $X_1 - X_2 = X_1^* - X_2^* = [3, -3]^T$. The linear constraints and concrete lower and upper bounds defining the range of the difference are as follows.

$$\delta_{x_1}^{1,2,\leq} = \delta_{x_1}^{1,2,\geq} = 3 \qquad \delta_{x_2}^{1,2,\leq} = \delta_{x_2}^{1,2,\geq} = -3 \qquad (x_1^1 - x_1^2) \in [3, 3] \qquad (x_2^1 - x_2^2) \in [-3, -3]$$

At the input layer, the abstract elements also track linear constraints and concrete bounds for variables x_1^1, x_2^1, x_1^2, and x_2^2. However, for this example, we primarily focus on constraints $\delta_{x_i}^{1,2,\geq}$ and $\delta_{x_i}^{1,2,\leq}$ and show the rest of the constraints in the Appendix 8.8. Next, we apply the affine transformer (defined in Section 4.1) to calculate bounds corresponding to $< x_3^1, x_3^2 >$ and $< x_4^1, x_4^2 >$. We show the derivation of linear constraints $\delta_{x_3}^{1,2,\geq}$ and $\delta_{x_3}^{1,2,\leq}$ below where $\delta_{x_1}^{1,2} = (x_1^1 - x_1^2)$ and $\delta_{x_2}^{1,2} = (x_2^1 - x_2^2)$. The symbolic bounds $\delta_{x_4}^{1,2,\geq}$ and $\delta_{x_4}^{1,2,\leq}$ are obtained similarly.

$$\delta_{x_3}^{1,2} = (x_1^1 - x_2^1) - (x_1^2 - x_2^2) \implies \delta_{x_3}^{1,2,\geq} = \delta_{x_3}^{1,2,\leq} = (x_1^1 - x_1^2) - (x_2^1 - x_2^2) = \delta_{x_1}^{1,2} - \delta_{x_2}^{1,2} \qquad (4)$$

To compute the concrete lower bound $\Delta_{lb}^{1,2,x_3}$ (or, upper bound) of $(x_3^1 - x_3^2)$ we substitute the concrete bounds of $\delta_{x_1}^{1,2}$ and $\delta_{x_2}^{1,2}$ in lower (upper) symbolic bounds of Eq. 4 for example:

$$\delta_{x_3}^{1,2,\leq} = \delta_{x_1}^{1,2} - \delta_{x_2}^{1,2} \implies \Delta_{lb}^{1,2,x_3} = \Delta_{lb}^{1,2,x_1} - \Delta_{ub}^{1,2,x_3} = 6$$

Next, we compute bounds corresponding to $< x_5^1, x_5^2 >$ by using the ReLU abstract transformer (formally introduced in Section 4.2) for the assignments $x_5^1 \leftarrow ReLU(x_3^1)$ and $x_5^2 \leftarrow ReLU(x_3^2)$. In this case, choices for the symbolic bounds are non-unique. Fig. 4a shows one of two possible choices for linear constraints $\delta_{x_5}^{1,2,\geq} = \delta_{x_3}^{1,2}$ and $\delta_{x_5}^{1,2,\leq} = 0$. $\delta_{x_5}^{1,2,\geq} = x_5^{1,\geq} - x_5^{2,\leq}$ and $\delta_{x_5}^{1,2,\leq} = x_5^{1,\leq} - x_5^{2,\geq}$ are alternative candidates. However, in the abstract domain, we only allow only one choice for $\delta_{x_5}^{1,2,\geq}$ and one choice for $\delta_{x_5}^{1,2,\leq}$ so we greedily select one of two possible candidates for both $\delta_{x_5}^{1,2,\geq}$ and $\delta_{x_5}^{1,2,\leq}$. For both choices, we first evaluate the concrete bounds of $(x_5^1 - x_5^2)$ by substituting all variables in the symbolic lower (or upper) bound with their respective concrete bounds and then pick the candidate with the more precise concrete bound. For example, the choice $\delta_{x_5}^{1,2,\geq} = \delta_{x_3}^{1,2}$ yields concrete bound $\Delta_{ub}^{1,2,x_5} = 6.0$ which is more precise than $\Delta_{ub}^{1,2,x_5} = 20.625$ calculated from $\delta_{x_5}^{1,2,\geq} = x_5^{1,\geq} - x_5^{2,\leq}$. Thus, we select $\delta_{x_5}^{1,2,\geq} = \delta_{x_3}^{1,2}$. Finally, we obtain bounds corresponding to $< x_7^1, x_7^2 >$ and $< x_8^1, x_8^2 >$ by applying the affine abstract transformer. We show concrete bounds for the difference of each pair of variables $(x_i^1 - x_i^2)$ in Fig. 3 and detailed analysis in Appendix 8.8.

3.1.7 Back-substitution for Concrete Bounds. We obtain the concrete bounds of each $(x_i^1 - x_i^2)$ by the back-substitution strategy used in most of the popular non-relational DNN verifiers e.g. CROWN [92], DeepPoly [69], α-CROWN [86], etc. In back-substitution, we start with the symbolic bounds $\delta_{x_i}^{a,b,\geq}$ (or, $\delta_{x_i}^{a,b,\leq}$) of $(x_i^1 - x_i^2)$ and then obtain concrete bounds Δ_{ub}^{a,b,x_i} (or, Δ_{lb}^{a,b,x_i}) of $(x_i^1 - x_i^2)$ by substituting concrete bounds of all the variables in $\delta_{x_i}^{a,b,\geq}$ (or, $\delta_{x_i}^{a,b,\leq}$). Commonly, back-substitution does not stop after a single concrete substitution step rather it refines Δ_{ub}^{a,b,x_i} (or, Δ_{lb}^{a,b,x_i}) by a sequence of steps with each step including a symbolic substitution, where all the variables in $\delta_{x_i}^{a,b,\geq}$ (or, $\delta_{x_i}^{a,b,\leq}$) are replaced by the corresponding symbolic bounds, followed by a concrete substitution. Although back-substitution is computationally more expensive than a single concrete substitution step, it obtains more precise concrete bounds Δ_{ub}^{a,b,x_i} (or, Δ_{lb}^{a,b,x_i}) improving RaVeN's precision.

3.2 Using Analysis Bounds to Solve the UAP Verification Problem

We will now explain how RaVeN combines DiffPoly analysis with product DNN analysis to create the MILP formulation. Additionally, through our illustrative example, we will compare RaVeN's approach to state-of-the-art baseline methods like [40] and [88]. This comparison will demonstrate that while the baseline methods fall short in confirming the absence of a UAP in our example, our approach successfully verifies the non-existence of a UAP.

3.2.1 State-of-the-art DNN Relational Verifiers. [40] only analyzes the product DNN and uses the concrete bounds obtained independently for each execution to verify UAP robustness. This approach does not track any dependencies across executions and just leverages standard DNN local robustness verification of individual inferences. However, DeepZ analysis on the product DNN computes for input region ϕ_t^1 the lower bound of $C_1^T N_{ex}(X_1)$ is -13.25 and for ϕ_t^2 the lower bound of $C_2^T N_{ex}(X_2)$ is -31.44. Since the lower bounds of both $C_1^T N_{ex}(X_1)$ and $C_2^T N_{ex}(X_2)$ are less than 0 this method can not prove that UAP does not exist. Next, we focus on the state-of-the-art approach (referred to as I/O formulation in the rest of the paper) for UAP verification introduced by [88]. The I/O formulation initially applies non-relational DNN verifiers (e.g., DeepZ) to the product DNN. Based on DeepZ analysis, for each execution, it extracts linear constraints connecting output variables to input variables. Lastly, it translates the cross-execution input constraints into linear constraints, represents the output specification Ψ as a MILP objective, and employs standard MILP solvers to find the optimal solution (detailed formulation in Appendix 9.1). For our illustrative example, the I/O formulation can only prove the absence of a UAP when the MILP solution is non-negative. However, the optimal MILP solution in this case is $-5.306 < 0$, highlighting that the I/O formulation lacks the precision to verify the relational property. This imprecision arises because the I/O formulation, while tracking dependencies at the input layers, neglects subsequent hidden layers, leading to a loss of precision.

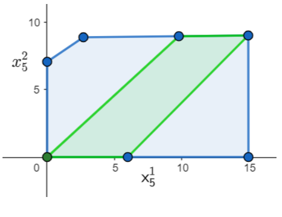


Fig. 5. For the variables x_5^1 and x_5^2 the convex region (green) obtained with constraints from DiffPoly analysis is more precise than the convex region (blue) formed without the difference constraints.

3.2.2 RaVeN MILP Formulation. We introduce a two-step enhancement to the MILP encoding in comparison to I/O formulation (same MILP objective) using our tool, RaVeN. To begin with, we relate the output of each layer to the output of the preceding layer by employing a set of linear constraints, commencing from the input layer. We replace non-linear activation layers (e.g., ReLU, Sigmoid, etc.) with convex overapproximations using concrete bounds obtained from DeepZ analysis, such as triangle relaxation [70] for ReLU. RaVeN's layerwise approach effectively captures linear dependencies across executions at the hidden layers, yielding an improved optimal solution of −1.564 compared to the I/O formulation (details behind this improvement in Appendix 9.2). Nonetheless, it remains insufficient for verifying the absence of UAP. In this case, the issue lies in the isolated computation of convex overapproximations for non-linear activation functions, which disregards the inter-dependencies between executions. To address this limitation, RaVeN utilizes the DiffPoly analysis and incorporates DiffPoly's custom abstract transformers for non-linear activation functions defined over pairs of executions. This approach computes convex overapproximations that consider inter-dependencies between execution pairs. Figure 5 illustrates this enhancement, showing how constraints derived from the DiffPoly analysis enhance the precision of the convex region at the hidden layers. The addition of the difference constraints from the DiffPoly analysis to the layerwise formulation of RaVeN improves the optimal value to 0 thereby proving the absence of UAP in the illustrative example. It is important to note that RaVeN employs the same MILP encoding for Ψ as utilized in the I/O formulation. The observed improvement is the result of RaVeN's enhanced capability in capturing the linear dependencies between outputs from multiple executions. The detailed MILP formulation for RaVeN is in Appendix 9.3.

4 RAVEN ALGORITHM

In this section, we present RaVeN's pseudocode, discuss its key components, and assess its asymptotic runtime. We provide a sketch of the soundness proofs of RaVeN in Section 4.7 with detailed proofs in Appendix 13. We first formally introduce the product DNN.

Definition 4.1 (Product DNN). Given any l layer DNN $N : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_l}$ and input specifiction Φ defined over k executions of N the product DNN $\mathcal{N}^k : \mathbb{R}^{n_0 \times k} \rightarrow \mathbb{R}^{n_l \times k}$ defined as sequential composition of l functions $\mathcal{N}_i^k : \mathbb{R}^{n_{i-1} \times k} \rightarrow \mathbb{R}^{n_i \times k}$ where $\mathcal{N}_i^k((X_1^i, \ldots, X_k^i)) = [N_i(X_1^i), \ldots, N_i(X_k^i)]^T$, for all $j \in [k]$. $X_j^i \in \mathbb{R}^{n_{i-1}}$ and $N_i : \mathbb{R}^{n_{i-1}} \rightarrow \mathbb{R}^{n_i}$ is the i-th layer of N.

Algorithm 1 shows the pseudocode for RaVeN. For the product DNN, an existing non-relational verifier (e.g. DeepZ) is used to obtain the concrete bounds for the outputs of all k executions at all layers, say $\mathscr{A}^k$ (line 5). We use the concrete bounds from product DNN analysis (line 7) to initialize DiffPoly analysis for all $\kappa = \binom{k}{2}$ pair of executions (line 8). Next, DiffPoly computes the symbolic and concrete bounds (denoted as $\mathscr{A}_\delta^{a,b}$) of the outputs and their differences w.r.t each pair of executions (line 8). Note that aside from handling differences, DiffPoly also maintains symbolic bounds on the variables from the product DNN that are relevant to the pair of executions it is analyzing. This allows DiffPoly to calculate the concrete bounds of these product DNN variables using back-substitution although DiffPoly can also be run independently from product DNN analysis. However, we decide to utilize the concrete bounds from the product DNN analysis, as they can be more precise compared to the bounds obtained by DiffPoly. Furthermore, this approach enables DiffPoly to benefit from any improvements made in the product DNN analysis. We produce linear constraints for all layers by utilizing the symbolic and concrete bounds obtained from DiffPoly analysis on all κ pairs of executions. (line 10). After layerwise linear constraints computation, we encode Ψ, as a MILP objective (line 11). Finally, we invoke a MILP solver on the MILP formulated using the linear constraints and MILP objective function to verify the relational verification problem (line 12). Note, Algorithm 1 shows a sequential implementation of RaVeN. However, we can parallelly run existing DNN abstract interpreters on each of k copies of N and parallelly execute DiffPoly interpreter on all $\binom{k}{2}$ difference networks. Next, we formally define the building blocks of RaVeN algorithm: DiffPoly domain and layerwise MILP formulation.

4.1 DiffPoly Abstract Domain

Next, we formally introduce the DiffPoly domain and the corresponding abstract transformers for the affine and activation (ReLU, Sigmoid, Tanh, etc.) assignments. For a list of $2n$ variables

Algorithm 1 RaVeN Algorithm

```
procedure RaVeN($\Phi, \Psi, N$)
    Input: $\Phi : \mathbb{R}^{n_0 \times k} \to \{true, false\}, \Psi : \mathbb{R}^{n_l \times k} \to \{true, false\}, N : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l}$.
    Verify: $\forall X_1, \dots, X_k \in \mathbb{R}^{n_0}.\ \Phi(X_1, \dots, X_k) \implies \Psi(N(X_1), \dots, N(X_k))$.
    $\mathcal{N}^k \leftarrow$ ConstructProductDNN($N, \Phi$)
    $\mathscr{A}^k \leftarrow$ ProdDNNAnalyzer($\mathcal{N}^k, \Phi, \mathcal{V}$)    ▹ $\mathcal{V}$ is existing non-relational DNN verifier
    for $a, b \in [k] \wedge a < b$ do
        $\mathcal{L}^a, \mathcal{U}^a, \mathcal{L}^b, \mathcal{U}^b \leftarrow$ExtractConcreteBounds($\mathscr{A}_i^k, a, b$)
        $\mathscr{A}_\delta^{a,b} \leftarrow$ DiffPolyExecutor($N^a, N^b, \Phi, \mathcal{L}^a, \mathcal{U}^a, \mathcal{L}^b, \mathcal{U}^b$)
    end for
    $\mathcal{M} \leftarrow$ [LayerwiseConstraints($\mathscr{A}_\delta^{a,b}, N, \Phi$) $\mid a, b \in [k] \wedge b < a$]    ▹ Constraints
    $\mathcal{M}^\Psi \leftarrow$RaVeNObjectiveFunction($\Psi$)    ▹ Objective Function Formulation
    return MILPSolver($\mathcal{M}, \mathcal{M}^\Psi$)    ▹ MILP Solver Invocation
end procedure
```

$[x_1^a, \dots, x_n^a]$, $[x_1^b, \dots, x_n^b]$ corresponding to a pair of execution of N the corresponding element in the DiffPoly domain $\mathcal{A}_{2n}$ is defined as $\overline{a} = [a_1, \dots, a_n]$. Here each a_i is associated with a pair of variables $< x_i^a, x_i^b >$. a_i associates (i) six symbolic bounds: symbolic lower and upper bounds for x_i^a, x_i^b and $(x_i^a - x_i^b)$ and (ii) six concrete bounds: concrete lower and upper bounds for x_i^a, x_i^b and $(x_i^a - x_i^b)$. We represent each a_i as a tuple $a_i =< C_{sym}^i, C_{con}^i >$ with C_{sym}^i and C_{con}^i denoting the symbolic and concrete bounds respectively:

$$C_{sym}^i = \{x_i^{a,\leq}, x_i^{b,\leq}, \delta_{x_i}^{a,b,\leq}, x_i^{a,\geq}, x_i^{b,\geq}, \delta_{x_i}^{a,b,\geq}\} \qquad C_{con}^i = \{l_{a,x_i}, l_{b,x_i}, \Delta_{lb}^{a,b,x_i}, u_{a,x_i}, u_{b,x_i}, \Delta_{ub}^{a,b,x_i}\}$$

The monotonic concretization function $\gamma_{2n} : \mathcal{A}_{2n} \to \wp(\mathbb{R}^{2n})$ mapping each abstract element $\overline{a}$ to the corresponding element in the concrete domain $\wp(\mathbb{R}^{2n})$ (powerset of $\mathbb{R}^{2n}$), is shown in Eq. 5 where for any $X \in \mathbb{R}^n$ we represent i-th coordinate of X as x_i.

$$\varphi_{2n}^\delta(X^a, X^b) = (X^a, X^b \in \mathbb{R}^n) \wedge (\forall i \in [n].(\delta_{x_i}^{a,b,\leq} \leq (x_i^a - x_i^b) \leq \delta_{x_i}^{a,b,\geq} \wedge \Delta_{lb}^{a,b,x_i} \leq (x_i^a - x_i^b) \leq \Delta_{ub}^{a,b,x_i}))$$
$$\varphi_n(X^a) = (X^a \in \mathbb{R}^n) \wedge (\forall i \in [n].(x_i^{a,\leq} \leq x_i^a \leq x_i^{a,\geq} \wedge l_{a,x_i} \leq x_i^a \leq u_{a,x_i}))$$
$$\gamma_{2n}(\overline{a}) = \{(X^a, X^b) \mid X^a, X^b \in \mathbb{R}^n \wedge \varphi_n(X^a) \wedge \varphi_n(X^b) \wedge \varphi_{2n}^\delta(X^a, X^b)\} \tag{5}$$

In the DiffPoly domain, for any deterministic function $f : \mathbb{R}^n \to \mathbb{R}^m$ the abstract transformer $T_f^\sharp : \mathcal{A}_{2n} \to \mathcal{A}_{2m}$ is required to satisfy the following soundness condition for all abstract elements $\overline{a} \in \mathcal{A}_{2n}$ where $T_f : \wp(\mathbb{R}^{2n}) \to \wp(\mathbb{R}^{2m})$ defines the corresponding concrete transformer

$$T_f(\gamma_{2n}(\overline{a})) \subseteq \gamma_{2m}(T_f^\sharp(\overline{a})) \quad \text{where } \forall \mathcal{X} \in \wp(\mathbb{R}^{2n}).\ T_f(\mathcal{X}) = \{(f(X), f(Y)) \mid (X, Y) \in \mathcal{X}\}$$

Next, we define abstract transformers for the DiffPoly domain.

4.2 DiffPoly ReLU Abstract Transformer

$ReLU : \mathbb{R} \to \mathbb{R}$ is defined as $ReLU(x) = \max(0, x)$. Let, $T_R^\sharp : \mathcal{A}_{2i} \to \mathcal{A}_{2i+2}$ be the abstract transformer that executes assignment statements $y_i^a \leftarrow ReLU(x_i^a)$, $y_i^b \leftarrow ReLU(x_i^b)$. For $\overline{a} = [a_1, \dots, a_i] \in \mathcal{A}_{2i}$, let $\overline{a'} = T_R^\sharp(a)$ represent the output of the transformer. First, for $\overline{a'} = [a'_1, \dots, a'_{i+1}]$, we compute the symbolic bounds $C_{sym}^{\prime j}$ for each a'_j where $j \in [i{+}1]$. In this case, for all $j \in [i].a'_j = a_j$ and a'_{i+1} is associated with the variable pair $< y_i^a, y_i^b >$. Since $ReLU$ is piecewise linear, we separately analyze cases where $ReLU$ acts as a linear function and cases where it demonstrates non-linear

Table 1. DiffPoly ReLU Cases

Cases from x_i^a	$x_-^{a,i} = (u_{a,x_i} \leq 0)$	$x_+^{a,i} = (l_{a,x_i} \geq 0)$	$x_\pm^{a,i} = \neg x_-^{a,i} \wedge \neg x_+^{a,i}$
Cases from x_i^b	$x_-^{b,i} = (u_{b,x_i} \leq 0)$	$x_+^{b,i} = (l_{b,x_i} \geq 0)$	$x_\pm^{b,i} = \neg x_-^{b,i} \wedge \neg x_+^{b,i}$
Cases from $\delta_{x_i}^{a,b}$	$\delta_-^i = (\Delta_{ub}^{a,b,x_i} \leq 0)$	$\delta_+^i = (\Delta_{lb}^{a,b,x_i} \geq 0)$	$\delta_\pm^i = \neg\delta_-^i \wedge \neg\delta_+^i$

behavior. Table 1 summarizes the separate cases we consider while designing the abstract transformer for *ReLU*. In Table 1, for any variable v, v_+ (or, v_-) denotes the case when values taken by v are always positive (or negative) and $v_\pm$ denotes the case when v can be both positive and negative.

Symbolic Bounds for $(y_i^a - y_i^b)$. We first consider cases where at least one of $ReLU(x_i^a)$ or $ReLU(x_i^b)$ behaves as a linear function and separately consider the case where both of them are non-linear. Similarly, we consider 3 scenarios based on the concrete bounds of $\delta_{x_i}^{a,b} = x_i^a - x_i^b$ (shown in Fig. 4) where we characterize the convex region having a minimum area that captures all possible values of $(y_i^a - y_i^b)$. In Table 2, we show the computation of the symbolic bounds for $(y_i^a - y_i^b)$ based on the cases for x_i^a and x_i^b. The first column shows the case, the second column shows the symbolic expression for $(y_i^a - y_i^b)$, and the last column shows its symbolic bounds. For the first four cases, $ReLU(x_i^a) - ReLU(x_i^b)$ behaves as a linear function and therefore our symbolic bounds are exact. For the remaining 5 cases, we compute symbolic bounds for $(y_i^a - y_i^b)$ overapproximating the exact values based on the symbolic bounds of y_i^a, y_i^b, x_i^a and x_i^b. We also consider 3 separate cases depicted in Table 3 (and in Fig 4) based on concrete bounds of $(x_i^a - x_i^b)$ where $\delta_{y_i}^{a,b,\leq}$ and $\delta_{y_i}^{a,b,\geq}$ are linear function of $\delta_{x_i}^{a,b} = (x_i^a - x_i^b)$. The cases described above are not mutually exclusive, resulting in multiple symbolic bound choices for $(y_i^a - y_i^b)$. However, in DiffPoly, we only allow a single symbolic upper bound and a lower bound for $(y_i^a - y_i^b)$. To resolve this, as described in

Table 2. Computation of the symbolic bounds for $\delta_{y_i}^{a,b}$ based on cases for x_i^a and x_i^b.

Case	$\delta_{y_i}^{a,b}$	Symbolic bounds $\delta_{y_i}^{a,b,\leq}$ and $\delta_{y_i}^{a,b,\geq}$
$x_-^{a,i} \wedge x_-^{b,i}$	0	$(\delta_{y_i}^{a,b,\leq} = 0) \wedge (\delta_{y_i}^{a,b,\geq} = 0)$
$x_+^{a,i} \wedge x_+^{b,i}$	$x_i^a - x_i^b$	$(\delta_{y_i}^{a,b,\leq} = \delta_{x_i}^{a,b}) \wedge (\delta_{y_i}^{a,b,\geq} = \delta_{x_i}^{a,b})$
$x_+^{a,i} \wedge x_-^{b,i}$	x_i^a	$(\delta_{y_i}^{a,b,\leq} = x_i^a) \wedge (\delta_{y_i}^{a,b,\geq} = x_i^a)$
$x_-^{a,i} \wedge x_+^{b,i}$	$-x_i^b$	$(\delta_{y_i}^{a,b,\leq} = -x_i^b) \wedge (\delta_{y_i}^{a,b,\geq} = -x_i^b)$
$x_\pm^{a,i} \wedge x_-^{b,i}$	$ReLU(x_i^a)$	$(\delta_{y_i}^{a,b,\leq} = y_i^{a,\leq}) \wedge (\delta_{y_i}^{a,b,\geq} = y_i^{a,\geq})$
$x_-^{a,i} \wedge x_\pm^{b,i}$	$-ReLU(x_i^b)$	$(\delta_{y_i}^{a,b,\leq} = -y_i^{b,\geq}) \wedge (\delta_{y_i}^{a,b,\geq} = -y_i^{b,\leq})$
$x_\pm^{a,i} \wedge x_+^{b,i}$	$ReLU(x_i^a) - x_i^b$	$(\delta_{y_i}^{a,b,\leq} = y_i^{a,\leq} - x_i^b) \wedge (\delta_{y_i}^{a,b,\geq} = y_i^{a,\geq} - x_i^b)$
$x_+^{a,i} \wedge x_\pm^{b,i}$	$x_i^a - ReLU(x_i^b)$	$(\delta_{y_i}^{a,b,\leq} = x_i^a - y_i^{b,\geq}) \wedge (\delta_{y_i}^{a,b,\geq} = x_i^a - y_i^{b,\leq})$
$x_\pm^{a,i} \wedge x_\pm^{b,i}$	$ReLU(x_i^a) - ReLU(x_i^b)$	$(\delta_{y_i}^{a,b,\leq} = y_i^{a,\leq} - y_i^{b,\geq}) \wedge (\delta_{y_i}^{a,b,\geq} = y_i^{a,\geq} - y_i^{b,\leq})$

Table 3. Computation of the symbolic bounds for $\delta_{y_i}^{a,b}$ based on cases for $(x_i^a - x_i^b)$.

Case	Symbolic bounds $\delta_{y_i}^{a,b,\leq}$ and $\delta_{y_i}^{a,b,\geq}$ for ReLU activation
δ_+^i	$(\delta_{y_i}^{a,b,\leq} = 0) \wedge (\delta_{y_i}^{a,b,\geq} = \delta_{x_i}^{a,b})$
δ_-^i	$(\delta_{y_i}^{a,b,\leq} = \delta_{x_i}^{a,b}) \wedge (\delta_{y_i}^{a,b,\geq} = 0)$
$\delta_\pm^i$	$(\delta_{y_i}^{a,b,\leq} = \lambda_{lb}^\delta \cdot \delta_{x_i}^{a,b} + \mu_{lb}^\delta) \wedge (\delta_{y_i}^{a,b,\geq} = \lambda_{ub}^\delta \cdot \delta_{x_i}^{a,b} + \mu_{ub}^\delta)$ with $\lambda_{ub}^\delta = \frac{\Delta_{ub}^{a,b,x_i}}{\Delta_{ub}^{a,b,x_i} - \Delta_{lb}^{a,b,x_i}}, \lambda_{lb}^\delta = -\frac{\Delta_{lb}^{a,b,x_i}}{\Delta_{ub}^{a,b,x_i} - \Delta_{lb}^{a,b,x_i}}, -\mu_{ub}^\delta = \mu_{lb}^\delta = \frac{\Delta_{lb}^{a,b,x_i} \times \Delta_{ub}^{a,b,x_i}}{\Delta_{ub}^{a,b,x_i} - \Delta_{lb}^{a,b,x_i}}$

Section 3, we greedily select the symbolic bounds that yield more precise concrete bounds based on concrete substitution (see Eq. 7). For example, consider the case specified by $(x_{\pm}^{a,i} \wedge x_{\pm}^{b,i} \wedge \delta_{+})$ there are two choices for $\delta_{y_i}^{a,b,\geq} = y_i^{a,\geq} - y_i^{b,\leq}$ and $\delta_{y_i}^{a,b,\geq} = \delta_{x_i}^{a,b}$. Let, $S_c(y_i^{a,\geq} - y_i^{b,\leq})$ and $S_c(\delta_{x_i}^{a,b})$ be their respective concrete upper bounds. Then we pick $\delta_{y_i}^{a,b,\geq} = y_i^{a,\geq} - y_i^{b,\leq}$ if $S_c(y_i^{a,\geq} - y_i^{b,\leq}) < S_c(\delta_{x_i}^{a,b})$ otherwise select $\delta_{y_i}^{a,b,\geq} = \delta_{x_i}^{a,b}$. Next, we discuss symbolic bound computation for y_i^a and y_i^b.

Symbolic Bounds for y_i^a and y_i^b. For cases $x_{-}^{a,i}$ and $x_{+}^{a,i}$ where the *ReLU* behaves like a linear function, the symbolic bounds for y_i^a can be directly expressed as a linear function of x_i^a. However, for the case, $x_{\pm}^{a,i}$ the *ReLU* function is no longer linear and we apply the linear relaxation [69, 92] to obtain the symbolic bounds of y_i^a using the concrete bounds l_{a,x_i} and u_{a,x_i}. The details are in the Appendix (Fig. 13). Bounds for y_i^b are derived similarly.

Concrete Bounds for y_i^a , y_i^b. We get concrete bounds for y_i^a , y_i^b from the product DNN execution.

Concrete Bounds for $(y_i^a - y_i^b)$. For $(y_i^a - y_i^b)$, we find concrete bounds using *back-substitution*. Each *back-substitution* step recursively applies symbolic substitution (Eq. 6) followed by concrete substitution (Eq. 7) to generate a set of possible candidates for concrete bounds and picks the most precise one. We provide a pseudo-code of the back-substitution algorithm in Appendix 12. For any variable δ, its symbolic upper bound $\delta^{\geq} = \overline{v_0} + \sum_i \overline{w_i} \cdot \overline{v_i}$ and symbolic lower bound $\delta^{\leq} = \underline{v_0} + \sum_i \underline{w_i} \cdot \underline{v_i}$, the symbolic substitutions $S_s(\delta^{\geq})$, $S_s(\delta^{\leq})$ and concrete substitutions $S_c(\delta^{\geq})$, $S_c(\delta^{\leq})$ are shown below. Here, $\overline{v_0}, \underline{v_0} \in \mathbb{R}$ and $\overline{v_i}^{\geq}, \overline{v_i}^{\leq}$, $\underline{v_i}^{\geq}, \underline{v_i}^{\leq}$ are symbolic bounds of variables, $\overline{v_i}^{lb}, \overline{v_i}^{ub}$, $\underline{v_i}^{lb}, \underline{v_i}^{ub}$ are the respective concrete bounds and $\overline{w_i}^{+} = \max(0, \overline{w_i})$, $\overline{w_i}^{-} = \min(0, \overline{w_i})$, $\underline{w_i}^{+} = \max(0, \underline{w_i})$, $\underline{w_i}^{-} = \min(0, \underline{w_i})$. Note, both symbolic and concrete substitutions for upper and lower bounds satisfy that $(S_s(\delta^{\geq}) \geq \delta) \wedge (S_c(\delta^{\geq}) \geq \delta)$ and $(S_s(\delta^{\leq}) \leq \delta) \wedge (S_c(\delta^{\leq}) \leq \delta)$.

$$S_s(\delta^{\geq}) = \overline{v_0} + \sum_i \overline{w_i}^{+} \cdot \overline{v_i}^{\geq} + \sum_i \overline{w_i}^{-} \cdot \overline{v_i}^{\leq} \qquad S_s(\delta^{\leq}) = \underline{v_0} + \sum_i \underline{w_i}^{+} \cdot \underline{v_i}^{\leq} + \sum_i \underline{w_i}^{-} \cdot \underline{v_i}^{\geq} \tag{6}$$

$$S_c(\delta^{\geq}) = \overline{v_0} + \sum_i \overline{w_i}^{+} \cdot \overline{v_i}^{ub} + \sum_i \overline{w_i}^{-} \cdot \overline{v_i}^{lb} \qquad S_c(\delta^{\leq}) = \underline{v_0} + \sum_i \underline{w_i}^{+} \cdot \underline{v_i}^{lb} + \sum_i \underline{w_i}^{-} \cdot \underline{v_i}^{ub} \tag{7}$$

4.3 DiffPoly Abstract Transformer For Differentiable Activations

For any differentiable function $g : \mathbb{R} \rightarrow \mathbb{R}$, we define $T_g^{\sharp} : \mathcal{A}_{2i} \rightarrow \mathcal{A}_{2i+2}$ as the abstract transformer for the assignments $y_i^a \leftarrow g(x_i^a)$ and $y_i^b \leftarrow g(x_i^b)$. Both Sigmoid and Tanh, being differentiable everywhere, can be modeled via g. We use the lower bound and the upper bound on the derivative of g to compute the symbolic bounds of $(y_i^a - y_i^b)$. The concrete bounds of y_i^a and y_i^b are obtained from product DNN analysis while concrete bounds of $(y_i^a - y_i^b)$ are calculated by back-substitution.

Symbolic Bounds Computation: Let, $l_{g'}$ and $u_{g'}$ be the lower and upper bound of $g'(x)$ over the range $x \in [l, u]$ where $l = \min(l_{a,x_i}, l_{b,x_i})$ and $u = \max(u_{a,x_i}, u_{b,x_i})$. We consider three cases from the 3rd row of Table 1 and show the symbolic bounds of $(y_i^a - y_i^b)$ for all three cases in Table 4 (also depicted in Appendix Fig. 14). This formulation holds for any differentiable function g provided $l_{g'}$ and $u_{g'}$ are easy to compute. For Sigmoid and Tanh, the derivative $g'(x)$ has a closed form, and $g'(x)$ is maximum at $x = 0$ and decreases as x increases (or, decreases). So, $l_{g'}$ and $u_{g'}$ computation only takes constant time given values of l and u. For y_i^a and y_i^b, we use concrete bounds - $l_{a,x_i}, u_{a,x_i}, l_{b,x_i}, u_{b,x_i}$ and apply the linear relaxation from [92], which also extends to differentiable functions with a closed form of the differential.

4.4 DiffPoly Affine Abstract Transformer

We describe the affine abstract transformer $T_A^{\sharp} : \mathcal{A}_{2i} \rightarrow \mathcal{A}_{2i+2}$ corresponding to the assignment statements $x_{i+1}^a \leftarrow v + \sum_{j=1}^{i} w_j \cdot x_j^a$ and $x_{i+1}^b \leftarrow v + \sum_{j=1}^{i} w_j \cdot x_j^b$ where v and all w_j are real numbers.

Table 4. Computation of the symbolic bounds for $(y_i^a - y_i^b)$ based on cases for $(x_i^a - x_i^b)$.

Case	Symbolic bounds $\delta_{y_i}^{a,b,\leq}$ and $\delta_{y_i}^{a,b,\geq}$ for any differentiable activation g
δ_+^i	$(\delta_{y_i}^{a,b,\leq} = l_{g'} \cdot \delta_{x_i}^{a,b}) \wedge (\delta_{y_i}^{a,b,\geq} = u_{g'} \cdot \delta_{x_i}^{a,b})$
δ_-^i	$(\delta_{y_i}^{a,b,\leq} = u_{g'} \cdot \delta_{x_i}^{a,b}) \wedge (\delta_{y_i}^{a,b,\geq} = l_{g'} \cdot \delta_{x_i}^{a,b})$
$\delta_\pm^i$	$(\delta_{y_i}^{a,b,\leq} = \lambda_{lb}^\delta \cdot \delta_{x_i}^{a,b} + \mu_{lb}^\delta) \wedge (\delta_{y_i}^{a,b,\geq} = \lambda_{ub}^\delta \cdot \delta_{x_i}^{a,b} + \mu_{ub}^\delta)$ with $l_{g'} = \min_{x \in [l,u]} g'(x)$ and $u_{g'} = \max_{x \in [l,u]} g'(x)$ $\lambda_{ub}^\delta = \frac{u_{g'} \times \Delta_{ub}^{a,b,x_i} - l_{g'} \times \Delta_{lb}^{a,b,x_i}}{\Delta_{ub}^{a,b,x_i} - \Delta_{lb}^{a,b,x_i}}, \lambda_{lb}^\delta = \frac{l_{g'} \times \Delta_{ub}^{a,b,x_i} - u_{g'} \times \Delta_{lb}^{a,b,x_i}}{\Delta_{ub}^{a,b,x_i} - \Delta_{lb}^{a,b,x_i}}, -\mu_{ub}^\delta = \mu_{lb}^\delta = \frac{(u_{g'} - l_{g'}) \times \Delta_{lb}^{a,b,x_i} \times \Delta_{ub}^{a,b,x_i}}{\Delta_{ub}^{a,b,x_i} - \Delta_{lb}^{a,b,x_i}}$

In this case, the difference $(x_{i+1}^a - x_{i+1}^b)$ can represented as $(x_{i+1}^a - x_{i+1}^b) = \sum_{j=1}^{i} w_j \cdot (x_j^a - x_j^b)$. Since for affine assignments, x_{i+1}^a (and x_{i+1}^b) is a linear function over x_j^as (and x_j^bs), we can directly compute the linear constraints that represent the symbolic bounds. For $\overline{a} \in \mathcal{A}_{2i}$, let $\overline{a'} = [a'_1, \ldots, a'_{i+1}] = T_A^\sharp(\overline{a})$ where $\overline{a'} \in \mathcal{A}_{2i+2}$ and $\forall j \in [i]. \; (a_j = a'_j)$. We show the symbolic bounds corresponding to a'_{i+1} in Eq. 8. The product DNN analysis provides the concrete bounds of x_{i+1}^a and x_{i+1}^b while $\Delta_{lb}^{a,b,x_{i+1}}$ and $\Delta_{ub}^{a,b,x_{i+1}}$ are calculated by performing back-substitution on $\delta_{x_{i+1}}^{a,b,\leq}$ and $\delta_{x_{i+1}}^{a,b,\geq}$ respectively.

$$x_{i+1}^{a,\leq} = x_{i+1}^{a,\geq} = v + \sum_{j=1}^{i} w_j \cdot x_j^a \quad x_{i+1}^{b,\leq} = x_{i+1}^{b,\geq} = v + \sum_{j=1}^{i} w_j \cdot x_j^b \quad \delta_{x_{i+1}}^{a,b,\leq} = \delta_{x_{i+1}}^{a,b,\geq} = \sum_{j=1}^{i} w_j \cdot \delta_{x_j}^{a,b} \tag{8}$$

DiffPoly vs DeepPoly with Transformer for the Difference of Activations: In Section 3.1.5, we explain why the existing DeepPoly domain is not suited for difference-bound computation between the outputs of a pair of DNN executions. It is natural to ask whether the precision improvement in difference tracking achieved by DiffPoly can be replicated by just designing a new abstract transformer for the DeepPoly domain handling the following assignments $y_i^a \leftarrow \sigma(x_i^a)$, $y_i^b \leftarrow \sigma(x_i^b)$ and $(y_i^a - y_i^b) \leftarrow \sigma(x_i^a) - \sigma(x_i^b)$ where $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ is the non-linear activation function. In this case, the DeepPoly domain lacks concrete, symbolic bounds on the difference $(x_i^a - x_i^b)$ and can only use the concrete, symbolic bounds of the individual variables x_i^a, x_i^b. This results in imprecise concrete bounds Δ_{lb}^{a,b,y_i} and Δ_{ub}^{a,b,y_i} of $(y_i^a - y_i^b)$ which in turn results in imprecise symbolic bounds (Table 3 and 4 uses the sign of the concrete bounds of difference for selecting the symbolic bounds). For instance, in the illustrative example, the symbolic upper bound of $(\delta_{x_5}^{1,2})$ with DeepPoly bounds results in concrete upper bound $\Delta_{ub}^{1,2,x_5} = 20.625$ while DiffPoly produces more precise concrete upper bound $\Delta_{ub}^{1,2,x_5} = 6.0$. Overall DiffPoly is more general and can precisely handle bivariate non-linear functions such as $\sigma(x) - \sigma(y)$ with inputs x, y coming from two distinct copies of the network. Furthermore, we demonstrate in Appendix 14.5 that DiffPoly can be expanded to encompass any linear combination of variables from k executions. This makes DiffPoly the first domain capable of computing precise bounds (both concrete and symbolic) of any linear combination of DNN outputs at each layer coming from different related executions.

4.5 RaVeN's Layerwise Constraint Formulation

In this section, we formally introduce RaVeN's layerwise constraint formulation. Consider $\mathscr{A}_\Delta = [\mathscr{A}_\delta^1, \ldots, \mathscr{A}_\delta^\kappa]^T$, that stores the symbolic and concrete bounds computed by all κ DiffPoly analyses, with $\mathscr{A}_\delta^j$ representing the bounds computed by the j-th analysis. RaVeN's constraint formulation algorithm takes as input $\mathscr{A}_\Delta$, network $N : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_l}$, and the input specification Φ and generates a set of linear constraints for each layer. Let, $\mathcal{L}^i$ represent the set of linear constraints over the outputs of the i^{th} layer, defining the convex region $\mathcal{L}_t^i \subseteq \mathbb{R}^{n_i \times k}$. In this case, $\mathcal{L}_t^i$ contains all possible outputs at i-th layer for all k executions. We compute $\mathcal{L}^i$ by adding linear constraints for all n_i variables at the i-th layer for each pair of executions using the concrete and symbolic bounds from the DiffPoly analysis for that pair. For instance, consider $a \in [k] \land b \in [k] \land (a < b)$, which defines a pair of executions. Here, $[x_1^a, \dots, x_{n_i}^a]$ and $[x_1^b, \dots, x_{n_i}^b]$ represent variables at the i-th layer for the pair of executions (a, b). Then the linear constraints added for this pair of executions are as follows where $j \in [n_i]$ and the concrete and symbolic bounds are from the DiffPoly analysis which in turn inherits the concrete bounds $l_{a,x_j}, u_{a,x_j}, l_{b,x_j}, u_{b,x_j}$ from product DNN analysis:

$$\begin{array}{lll} x_j^{a,\le} \le x_j^a \le x_j^{a,\ge} & x_j^{b,\le} \le x_j^a \le x_j^{b,\ge} & \delta_{x_j}^{a,b,\le} \le (x_j^a - x_j^b) \le \delta_{x_j}^{a,b,\ge} \\ l_{a,x_j} \le x_j^a \le u_{a,x_j} & l_{b,x_j} \le x_j^a \le u_{b,x_j} & \Delta_{lb}^{a,b,x_j} \le (x_j^a - x_j^b) \le \Delta_{ub}^{a,b,x_j} \end{array} \tag{9}$$

In Eq. 9, the third column illustrates the additional difference constraints added for a variable pair, while the remaining constraints constitute RaVeN's layerwise formulation, as elaborated in Section 3.2.2. Note that, as discussed earlier, in DiffPoly analysis, up to two valid symbolic lower or upper bounds can be generated for each variable and their difference. For efficiency in concrete bounds computation with back-substitution, DiffPoly restricts to a single symbolic lower and upper bound. However, in the MILP formulation, all valid bounds are incorporated. The input specification Φ, defined as a conjunction of linear constraints over the inputs, is directly encoded as a set of linear constraints $\mathcal{L}^0$ at the input layer. The linear constraints for all l layers are then generated by aggregating layerwise constraints $\mathcal{L}^i$ with input linear constraints $\mathcal{L}^0$.

4.6 RaVeN MILP Encoding

We provide the general encoding of Ψ as MILP objective for relational DNN specifications described in Section 2.1. We add the MILP encoding of Ψ to the layerwise constraints from Section 4.5 to formulate the MILP instance. Let $Y_1, \dots, Y_k$ be the DNN's output for k executions, for all $i \in [m]$ and $j \in [n]$, $x_{i,j}$ and z_i be integer variables and for all $i' \in [k]$, $C_{i,j,i'} \in \mathbb{R}^{n_l}$ where m is the number of clauses in Ψ and n is number of literals in each clause (see Section 2.1). Then the MILP objective is as follows

$$\min_{(Y_1,\dots,Y_k)} \sum_{i=1}^{m} z_i \quad \text{s.t.} \quad x_{i,j} = \psi_{i,j}(Y_1, \dots, Y_k) = \left(\sum_{i'=1}^{k} C_{i,j,i'}^T Y_{i'} \ge 0 \right); z_i = \left(\sum_{j=1}^{n} x_{i,j} \ge 0 \right) \tag{10}$$

The proof of the correctness of the MILP formulation is in Appendix 13.6. For the common properties (e.g. UAP, targeted-UAP, worst-case hamming distance, etc.) $m = k$, $n = n_l$ and the MILP objective introduces only $k \times (n_l + 1)$ integer variables where n_l is the output dimension of the DNN (Appendix 14.4). Hence irrespective of the size of the network, the number of integer variables only depends on the number of executions k and n_l which is in general a small constant (i.e. 10 for commonly used MNIST and CIFAR10 networks). Since the number of integer variables is the primary bottleneck of MILP optimization, RaVeN scales to large DNNs by only introducing a small number of integer variables (n_l+1) per execution. This differs from the naive MILP which introduces an integer variable at each activation and does not scale past even small networks containing a few hundred neurons. Besides decreasing the count of integer variables, RaVeN efficiently infers linear constraints for the MILP encoding that are sound while improving the precision of the over-approximated convex region (illustrated in Figure 5 of the paper). This requires - (i) recognizing that tracking the difference between the outputs of a pair of DNN executions helps in improving precision while maintaining scalability, and (ii) designing and leveraging DiffPoly analysis on $\binom{k}{2}$ pairs of executions while computing provably correct constraints across multiple executions.

4.7 Soundness Proof Sketch of RaVeN

In this section, we outline the soundness proof for various components of RaVeN. Detailed proofs are in Appendix 13. We start with the soundness proofs of all DiffPoly transformers.

4.7.1 Soundness of DiffPoly ReLU Tansformer. We first state the lemmas required to prove the soundness of $T_R^\sharp$. Proofs of all cases shown in Fig. 4, Lemma 4.2, and 4.3 are in Appendix 14.1.

Lemma 4.2. *(Correctness of symbolic bounds in Table 2 and 3) If* $x_i^a \in [l_{a,x_i}, u_{a,x_i}]$, $x_i^b \in [l_{b,x_i}, u_{b,x_i}]$ *and* $\delta_{x_i}^{a,b} = (x_i^a - x_i^b) \in [\Delta_{lb}^{a,b,x_i}, \Delta_{ub}^{a,b,x_i}]$ *and* $\delta_{y_i}^{a,b} = ReLU(x_i^a) - ReLU(x_i^b)$ *then* $\delta_{y_i}^{a,b,\leq} \leq \delta_{y_i}^{a,b} \leq \delta_{y_i}^{a,b,\geq}$ *where* $\delta_{y_i}^{a,b,\leq}$ *and* $\delta_{y_i}^{a,b,\geq}$ *defined in Table 2 and 3.*

Lemma 4.3. *(Correctness of concrete bounds computed by the ReLU transformer) If* $x_i^a \in [l_{a,x_i}, u_{a,x_i}]$, $x_i^b \in [l_{b,x_i}, u_{b,x_i}]$ *and* $\delta_{x_i}^{a,b} = (x_i^a - x_i^b) \in [\Delta_{lb}^{a,b,x_i}, \Delta_{ub}^{a,b,x_i}]$, $y_i^a = ReLU(x_i^a)$, $y_i^b = ReLU(x_i^b)$, $\delta_{y_i}^{a,b} = y_i^a - y_i^b$ *then* $l_{a,y_i} \leq y_i^a \leq u_{a,y_i}$, $l_{b,y_i} \leq y_i^b \leq u_{b,y_i}$, *and* $\Delta_{lb}^{a,b,y_i} \leq \delta_{y_i}^{a,b} \leq \Delta_{ub}^{a,b,y_i}$ *where* Δ_{lb}^{a,b,y_i} *and* Δ_{ub}^{a,b,y_i} *computed by applying back-substitution on* $\delta_{y_i}^{a,b,\leq}$ *and* $\delta_{y_i}^{a,b,\geq}$ *respectively.*

The concrete transformer $T_R : \wp(\mathbb{R}^{2i}) \longrightarrow \wp(\mathbb{R}^{2i+2})$ for the ReLU assignments $y_i^a \leftarrow ReLU(x_i^a)$, $y_i^b \leftarrow ReLU(x_i^b)$ is defined as $T_R(\mathcal{X}) = \{([x_1^a, \ldots, x_i^a, y_i^a]^T, [x_1^b, \ldots, x_i^b, y_i^b]^T) \mid (X^a, X^b) \in \mathcal{X}\}$ where $y_i^a = ReLU(x_i^a)$, $y_i^b = ReLU(x_i^b)$, $\mathcal{X} \subseteq \mathbb{R}^{2i}$ and $X^a = [x_1^a, \ldots, x_i^a]^T \in \mathbb{R}^i$, $X^b = [x_1^b, \ldots, x_i^b]^T \in \mathbb{R}^i$.

Theorem 4.4. *(Soundness of DiffPoly Relu Transformer) For any abstract element* $\overline{a} \in \mathcal{A}_{2i}$ $T_R(\gamma_{2i}(\overline{a})) \subseteq \gamma_{2i+2}(T_R^\sharp(\overline{a}))$.

Proof. The proof is in Appendix 13.1. □

4.7.2 Soundness of DiffPoly Differentiable Activation Transformer. Proof of all the cases from Table. 4 are in Appendix 14.2. Lemma 13.1 proves the soundness of the symbolic bounds, while Lemma 13.2 proves the soundness of concrete bounds. The comprehensive soundness proof for the DiffPoly's transformer for differentiable activations is in Appendix 13.2.

4.7.3 Soundness of DiffPoly Affine Transformer. Lemma 13.4 proves the soundness of the symbolic bounds corresponding to the DiffPoly affine transformer, while Lemma 13.5 proves the soundness of the corresponding concrete bounds. A comprehensive soundness proof for the DiffPoly affine transformer is in Appendix 13.3.

4.7.4 Soundness of Product DNN Analysis. We prove that the output region $\mathbb{P} \subseteq \mathbb{R}^{n_l \times k}$ obtained by running existing DNN abstract interpreters e.g. [68] on each of k copies of N contains all possible output w.r.t all k executions on inputs satisfying Φ. Let, $\forall i \in [k]$ $\phi_{in}^i : \mathbb{R}^{n_0} \longrightarrow \{true, false\}$ defines the L_∞ input region $\phi_t^i = \|X - X_i^*\|_\infty \leq \epsilon$ for each of k executions. Existing DNN abstract interpreters operate on these individual input regions ϕ_t^i and compute the overapproximated output region $\mathcal{P}_i \subseteq \mathbb{R}^{n_l}$ that satisfies $\forall X \in \mathbb{R}^{n_0}.\phi_{in}^i(X) \implies (N(X) \in \mathcal{P}_i)$. The output region $\mathbb{P} \subseteq \mathbb{R}^{n_l \times k}$ is the cross-product of all k output regions $\mathbb{P} = \bigtimes_{i=1}^{k} \mathcal{P}_i$. Now, we show that $\mathbb{P}$ contains all possible outputs of $\mathcal{N}^k(X)$ provided $X \in \mathbb{R}^{n_0} \times k$ satisfies Φ.

Theorem 4.5. *(Soundness of Product DNN analysis)* $\forall (X_1, \ldots, X_k) \in \mathbb{R}^{n_0 \times k}.\Phi((X_1, \ldots, X_k)) \implies (\mathcal{N}^k((X_1, \ldots, X_k)) \in \mathbb{P})$.

Proof. The proof is in Appendix 13.4. □

4.7.5 Soundness of RaVeN MILP Formulation. We prove that for all layer $i \in [l]$ the convex region $\mathcal{L}_t^i \subseteq \mathbb{R}^{n_i \times k}$ defined by the linear constraints $\mathcal{L}^i$ contain all possible outputs at i-th layer for all k executions. For the input region, we show $\Phi_t \subseteq \mathcal{L}_t^0$.

THEOREM 4.6. *(Soundness of Linear constraints)* $\Phi_t \subseteq \mathcal{L}_t^0$ *and* $\forall i \in [l].\forall X_1, \dots X_k \in \mathbb{R}^{n_0}$. $\Phi(X_1, \dots, X_k) \implies (N^i(X_1), \dots, N^i(X_k)) \in \mathcal{L}_t^i$ *where* $N^i : \mathbb{R}^{n_0} \to \mathbb{R}^{n_i}$ *is the composition of first i layers of the network N,* $N^i = N_1 \circ \cdots \circ N_i$.

PROOF. The proof is in Appendix 13.5. □

Asymptotic Runtime Analysis: We provide the runtime analysis of RaVeN in Appendix 13.7.

5 EVALUATION

We evaluate the effectiveness of RaVeN on a wide range of relational properties and a diverse set of neural networks and datasets. We consider the following relational properties: UAP, targeted UAP, hamming distance, and monotonicity as formally defined in Appendix 8.3. For UAP and Hamming Distance properties, we compare our method to the existing baselines highlighted above in Section 3. The first baseline we consider is individual verification (see Section 3.2.1) which is work by Khedr and Shoukry [40]. The second baseline is an instantiation of the work done by Zeng et al. [88] with state-of-the-art non-relational verifiers DeepZ [68] and DeepPoly [69] which we call I/O Formulation (see Section 3.2.1). For these properties, our experimental results indicate that RaVeN is always more precise than existing methods and can verify significantly more properties. For monotonicity, we compare our methods to two existing baselines Liu et al. [48] and Pasado [44].

5.1 Experimental Setup

Datasets. For UAP based experiments, we use the popular MNIST [45] and CIFAR10 [42] image datasets. We also use MNIST for the Hamming distance experiments. For our monotonicity experiments, we use the Boston Housing (BH) dataset [37] and the Adult dataset [8]. The BH dataset contains 12 housing attributes such as age, tax, rooms, etc. and the target is housing price. The Adult dataset contains 87 features such as age, education, marital status, etc.
Neural Networks. Table 5 shows the MNIST, CIFAR10, BH, and Adult neural network architectures used in our experiments. We use standard network architectures (Convolutional and Fully-connected) commonly seen in other neural network verification works [68, 69]. We consider networks trained with standard training, DiffAI [53], CROWN-IBP [90], projected gradient descent (PGD) [50], and a monotonicity training scheme [34].
Non-relational Verifier. We instantiate both RaVeN and I/O Formulation with either DeepPoly or DeepZ. Although RaVeN works with other non-relational verifiers including SOTA "Branch and Bound" based verifiers like α, β-CROWN [79] and MNBaB [28]. We use DeepPoly or DeepZ because they are fast and widely used for initializing complete verifiers. For example, α, β-CROWN uses CROWN (equivalent to DeepPoly). We also compare RaVeN's performance with α, β-CROWN and MNBaB in Section 5.6.
Implementation Details. We implemented our method in Python with Pytorch V1.11 and Gurobi V10.0.3 as an off-the-shelf MILP solver. Our MNIST experiments were performed on an Intel(R) Core(TM) i7-12800HX @ 4.80 GHz with 16 GB of memory and the remainder of our experiments on an Intel(R) Core(TM) i9-9900KS CPU @ 4.00GHz with 64 GB of memory. Unless otherwise specified, we use DeepZ [68] to perform bound analysis on the product DNN and use the same verifier for the baselines. We use Gurobi with a timeout of 5 minutes to solve MILP problems.

Table 5. Network Information and Runtime (s) averaged over ϵ values considered in this paper

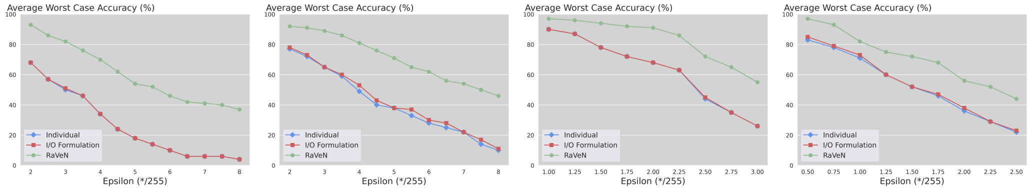
Dataset	Model	Type	Train	# Layers	# Params	Ind. Veri.	I/O Form.	RaVeN	MILP time
MNIST	IBP-Small	Conv	IBP	7	60k	0.04	0.12	1.98	1.01
	ConvSmall	Conv	DiffAI	7	80k	0.30	0.39	7.40	4.06
	IBP	Conv	IBP	9	400k	0.42	0.46	19.33	7.79
	ConvBig	Conv	DiffAI	13	1.8M	6.46	6.50	23.19	16.61
	Hamming	FC	PGD	3	39k	0.04	0.14	2.21	2.02
CIFAR10	IBP-Small	Conv	IBP	7	60k	0.29	0.47	8.39	5.03
	ConvSmall	Conv	DiffAI	7	80k	0.44	0.57	12.59	6.61
	IBP	Conv	IBP	9	2.2M	36.44	36.56	200.16	161.66
	ConvBig	Conv	DiffAI	13	2.5 M	16.19	16.29	185.05	161.63
Dataset	Model	Type	Train	# Layers	# Params	Liu et al.	Pasado	RaVeN	DiffPoly
BH	12x1	FC	Mono	3	312	0.25	×	-	0.02
Adult	10 x 10	FC	Standard	5	980	×	36.70	4.23	0.87

5.2 Relational Properties

The formal definitions for UAP, targeted UAP, and hamming distance given in Appendix 8.3 involve verifying that there does not exist an attack that can change all DNN predictions on a given input set by perturbing all the inputs with a single perturbation. While RaVeN can handle this problem, it is pessimistic and perturbations of this nature, although dangerous, rarely occur in reality. Instead, we bound the worst-case accuracy of the neural network under a UAP attack. Formally, we report a the verified worst-case accuracy which is a lower bound (as RaVeN is incomplete) on a^*, the true worst-case accuracy. For network N and inputs $X_1, \ldots X_k$ where $\forall v \in \mathbb{R}^{n_0}$ s.t. $||v||_p \leq \epsilon . \frac{1}{k} \sum_{i=1}^{k} (N(X_i + v) = Y_i) \geq a$ and Y_i is the correct label of X_i. Note that a result is better if it more tightly approximates a^* in this case since all presented methods are sound the best result is the one with the greatest value. For hamming distance, we perform a similar relaxation upper bounding the true worst case hamming distance. Thus, for hamming distance, smaller is better. For monotonicity, we are given a set of monotonic features and report the percentage of those features we can verify. For monotonicity, larger is better.

5.3 Universal Adversarial Perturbation Verification

We compare the performance of RaVeN vs the two baselines for worst-case accuracy under UAP attack on the MNIST and CIFAR10 networks. For each experiment, we verify a batch of 5 images. We repeat 20 times on randomly selected images, reporting the average worst-case accuracy. We use the standard ϵ values used in the literature [68, 69]. We additionally analyze RaVeN vs. baselines on the targeted UAP verification problem in Appendix 15.1.

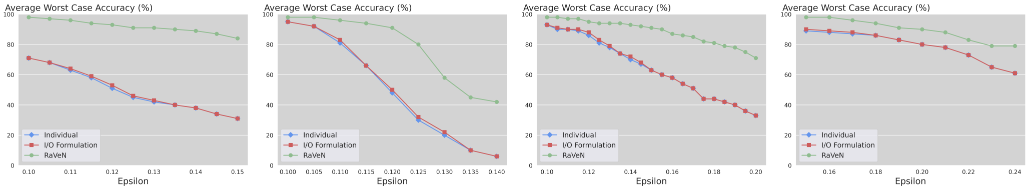


(a) IBP-Small (b) ConvSmall (c) IBP (d) ConvBig

Fig. 6. Average worst case UAP accuracy for convolutional networks trained on CIFAR10

5.3.1 Comparison on CIFAR10 Networks. Figure 6 compares the worst-case accuracy (%) on the CIFAR10 dataset with a variety of training methods (Crown-IBP, DiffAI) and network architectures

(IBP-Small, ConvSmall, IBP, ConvBig). We observe that RaVeN outperforms all baselines significantly for all networks, training methods, and ϵs. For example, we see that for IBP-Small trained with Crown-IBP that RaVeN obtains up to 25% higher verified average worst case UAP accuracy verified when compared to baselines on all ϵs. Note that all these cases I/O formulation hardly outperforms Individual verification. We observe similar results on the other networks.

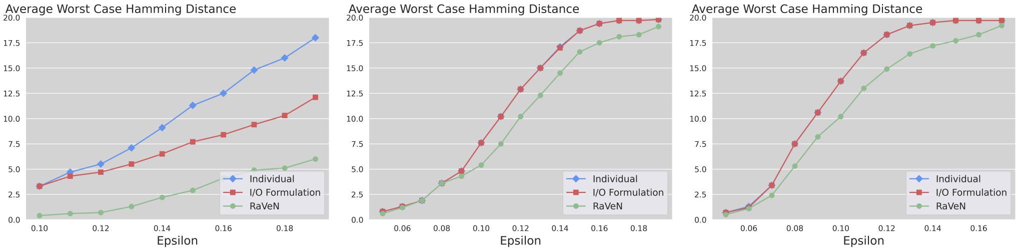


(a) IBP-Small (b) ConvSmall (c) IBP (d) ConvBig

Fig. 7. Average Worst case UAP accuracy for convolutional networks trained on MNIST

5.3.2 *Comparison on MNIST Networks.* Figure 7 shows similar results to CIFAR10 with the same diverse range of networks and training methods. Particularly, we observe that for IBP-Small RaVeN verifies an additional 25% accuracy when compared to baselines at $\epsilon = 0.1$. We observe that as ϵ grows RaVeN's relative benefit is greater, this is especially clear when for IBP (Figure 7 c).

5.3.3 *Runtime Analysis.* Table 5 shows the average runtime in seconds for each method. We observe that RaVeN time > I/O Formulation time > Independent Verification time. We note that even with more time the baseline approaches would not achieve any better results as they are limited and can not get more precise. Note that a majority of the time for RaVeN is taken by the MILP solver as seen in Table 5. As RaVeN is the first tool to show that cross-execution information aids in relational verification we believe runtime can be improved with future research. We also note that our timings are comparable to the timeouts given in the SOTA competition for verification of NNs (VNN-Comp [12]) (216 seconds per instance) even though we are verifying sets of 5 images.



(a) ReLU (b) Sigmoid (c) Tanh

Fig. 8. Average Worst Case Hamming Distance with different activation functions (smaller is better)

5.4 Hamming Distance Verification

We use MNIST as the base dataset and train a 3-layer fully connected network with 200 neurons in the hidden layers. We use a range of activation functions (ReLU, Tanh, Sigmoid). The network is adversarially trained with PGD to identify between classes 0 and 1. In this experiment, DeepPoly is used to instantiate both the baselines and RaVeN. Figure 8 shows the worst case hamming distance for strings of length 20 for different activation functions and ϵ values. For all ϵ values and string lengths, RaVeN outperforms both baselines, e.g. at $\epsilon = 0.3$ for Tanh the baselines obtain 20 and 19.85 while RaVeN obtains 15. We especially see that for Sigmoid and Tanh activations the baselines perform identically while RaVeN significantly outperforms both of them.

5.5 Monotonicity Verification

We verify the monotonicity of networks with both Tanh and ReLU activations trained on the Adult [8] and BH [37] datasets respectively. We compare our methods against the SOTA monotonicity verifier for Tanh networks, Pasado [44] using the Adult dataset with 5 monotonic features (same features as previous works [44, 66]). Monotonicity can be verified directly by DiffPoly without the need for any MILP formulation. For incomplete verifiers such as RaVeN, imprecisions accumulate during the analysis. By splitting the input region and verifying each region separately we can get a sound analysis which is sometimes more precise than the original analysis with some additional computation cost. Input splitting is a common tool used in other verification papers as a way to increase precision [38]. We use input splitting for monotonicity for two reasons: 1. the monotonic input specification only has one dimension of variation and is thus easy to split, and 2. DiffPoly/RaVeN verifies monotonicity very quickly in comparison to SOTA methods so we can split to gain precision while still having faster runtime. For both RaVeN and DiffPoly we split the input region 10 times before verifying. Figure 9 shows the results of RaVeN and DiffPoly compared to Pasado and its baselines (Zonotope, Interval). For small ϵ Pasado slightly outperforms RaVeN (92% vs 94%); however, as ϵ grows the benefit of RaVeN becomes clear (66% vs 2% at $\epsilon = 4$). We observe that DiffPoly alone can perform on par with Pasado while running significantly faster (0.87s vs 36.7s, while RaVeN sits in the middle at 4.23s). For ReLU networks we compare against Liu et al. [48] as Pasado is unable to handle ReLU (Liu et al. [48] only handles ReLU). We verify a single feature on the Boston Housing dataset over the 98 test images. Liu et al. [48] can verify all 98 inputs for monotonicity for each $\epsilon = [10, 20, 30]$. On the other hand, DiffPoly is able to verify $[96, 95, 95]$ inputs for $\epsilon = [10, 20, 30]$, but we note that DiffPoly is significantly faster (0.02s vs 0.25s). We observe that DiffPoly and RaVeN are powerful monotonicity verifiers that can handle a wider range of networks/activation functions than both baselines achieving good results in significantly less time.

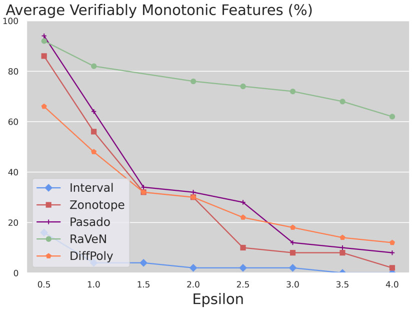
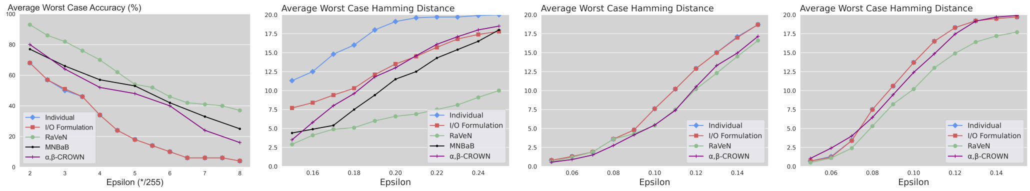


Fig. 9. Average % of Verified Monotonic Features on Adult Dataset

5.6 Ablation Studies

In this section, we show an ablation study comparing RaVeN to stronger individual verifiers: MNBaB [28] and α, β-CROWN [79]. We further show an ablation study on the benefits of adding difference constraints compared to only adding the layerwise formulation. In Appendix 15.2, we show RaVeN performs well compared to baselines when all of them use DeepPoly [69] instead of DeepZ [68].



(a) IBP-Small (CIFAR) (b) Hamming (ReLU) (c) Hamming (Sigmoid) (d) Hamming (Tanh)

Fig. 10. Comparison of RaVeN against MNBaB and α, β-CROWN

5.6.1 *Comparison to MNBaB and α, β-CROWN.*

MNBaB [28] and α, β-CROWN [79] use branching to obtain better precision at the cost of runtime. Although both MNBaB and α, β-CROWN are complete for non-relational properties for DNNs with piece-wise linear activations such as ReLU, they are imprecise for relational verification as they do not take the cross-execution constraints into account. Furthermore, both MNBaB and α, β-CROWN cannot verify monotonicity, whereas both DiffPoly and RaVeN can handle monotonicity. We instantiate MNBaB and α, β-CROWN with a 2-minute timeout per individual input. Note that although RaVeN is given a timeout of 5 minutes for MILP solving, for individual verifiers to perform UAP verification they must individually verify each input in the batch giving MNBaB and α, β-CROWN a total of 10 and 40 minutes for UAP and hamming distance verification respectively. Figure 10 compares RaVeN to MNBaB and α, β-CROWN on UAP verification for IBP-Small on CIFAR10 and for hamming distance verification on MNIST with different activations. Note that MNBaB does not currently support Sigmoid or Tanh activations. Similar to the above experiments, we instantiate RaVeN with DeepZ for IBP-Small and DeepPoly for hamming distance networks. We observe that RaVeN consistently performs better than MNBaB and α, β-CROWN (except for the hamming distance network with sigmoid activations for small ϵs). For example, for hamming distance with ReLU activations at $\epsilon = 0.25$, RaVeN can verify an average worst-case hamming distance of 10 while MNBaB and α, β-CROWN only obtain 18 and 18.5 respectively. For IBP-Small on CIFAR10 at $\epsilon = 8/255$, RaVeN can verify a worst-case UAP accuracy of 37% while MNBaB and α, β-CROWN only obtain 25% and 16% respectively.

In Table 6, we show a runtime comparison between RaVeN, MNBaB, and α, β-CROWN on the same networks as Figure 10. We observe that RaVeN takes less time than MNBaB and α, β-CROWN in all instances. Note that for Sigmoid and Tanh activations, α, β-CROWN is equivalent to α-CROWN [87] which does not support branching resulting in lower runtimes. In all instances, MNBaB and α, β-CROWN take significantly more time (> 37.7× more time for hamming distance with ReLU activations).

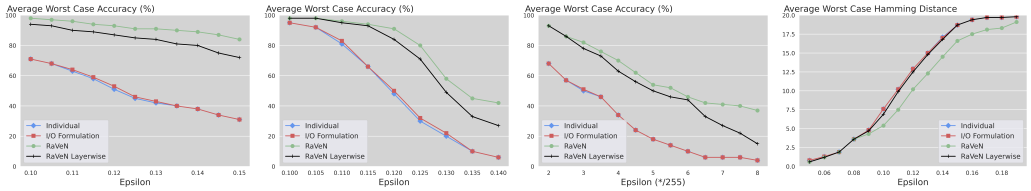
Table 6. Runtime Comparison (in secs) between RaVeN, MNBaB, and α, β-CROWN

Dataset	Model	Activation	RaVeN	MNBaB	α, β-CROWN
MNIST	Hamming	ReLU	4.92	209.38	185.91
	Hamming	Sigmoid	1.15	×	3.05
	Hamming	Tanh	2.37	×	5.77
CIFAR10	IBP-Small	ReLU	8.39	23.13	39.92
Adult	10 × 10	Tanh	4.23	×	×

5.6.2 Benefits of Difference Constraints. Figure 11 shows the benefits of adding difference constraints. In each example, RaVeN with difference constraints outperforms RaVeN layerwise without difference constraints. For example, for IBP-Small on CIFAR10 we see at $\epsilon = 8$ adding difference constraints increases the accuracy bound from 15% to 37%. The benefit of difference constraints is especially highlighted in the hamming distance example (d) as only by adding difference constraints is RaVeN able to outperform the baseline methods. A runtime comparison between RaVeN layerwise and RaVeN with difference constraints can be found in Appendix 15.3.

6 RELATED WORK

DNN Verifiers. Prior works in DNN verification [1] primarily focus on proving whether a DNN satisfies L_∞ robustness [69, 80] property. In this case, existing DNN verifiers show that all inputs inside a given L_∞ region [16] are properly classified. The DNN verifiers are broadly categorized into three main categories - (i) sound but incomplete verifiers which may not always prove property even if it holds [31, 63, 67–69, 86, 87], (ii) complete verifiers that can always prove the property if it holds [5, 13, 14, 25, 28, 30, 64, 71, 78, 79, 91] and (iii) verifiers with probabilistic guarantees



(a) IBP-Small (MNIST) (b) ConvSmall (MNIST) (c) IBP-Small (CIFAR10) (d) Hamming (Sigmoid)

Fig. 11. Comparison of RaVeN with difference constraints with RaVeN with only layerwise formulation.

[19]. However, all of these verifiers verify properties defined over single DNN execution and are ineffective for verifying interesting relational properties [17] such as UAP verification [88] and monotonicity [74] defined over multiple DNN executions.

DNN Relational Verifiers. Existing DNN relational verifiers can be grouped into two main categories - (i) verifiers for relational properties (UAP, monotonicity, etc.) defined over multiple executions of the same DNN, [40, 88], (ii) verifiers for relational properties (local DNN equivalence [58]) defined over multiple executions of different DNN on the same input [58, 59]. For relational properties defined over multiple executions of the same DNN the existing verifiers [40] reduce the verification problem into L_∞ robustness problem by constructing product DNN with multiple copies of the same DNN. However, the relational verifier in [40] treats all k executions of the DNN as independent and loses precision. The state-of-the-art DNN relational verifier [88] although tracks the relationship between inputs used in multiple executions at the input layer, does not track the relationship between the inputs fed to the subsequent hidden layers and can only achieve a marginal improvement over the baseline verifiers that treat all executions independently. ITNE [81] is a verifier for global robustness based on difference tracking. Global robustness measures the largest change to the output of a single class over the entire dataset (local robustness lifted to the dataset) whereas the UAP property considered in this work focuses on the number of points a single perturbation can cause to misclassify over a set of inputs which can be from different classes. Furthermore, RaVeN is more precise (Eq. 6 in [81] is covered by Table 2, RaVeN gains precision by also considering the constraints in Table 3) and handles more activations than ITNE.

Relational Verification of Programs. Compared to DNNs, significantly more work exist for verifying different relational properties, such as information flow security, determinism, etc. on programs [7, 9, 11, 15, 18, 26, 27, 29, 41, 65, 73, 77]. Standard programs and DNNs have different computational structure. For example, programs have loops while DNNs have a large number of non-linear activations. These structural differences create specific challenges for the relational verification of DNNs not seen for programs and vice-versa.

7 CONCLUSION

In this work, we developed a new framework called RaVeN to verify the relational properties of DNNs based on our novel approach of difference tracking with the DiffPoly abstract domain. We run extensive experiments on multiple relational properties including UAP verification, monotonicity, etc., and show that RaVeN outperforms the state-of-the-art relational verifier [88] on all of them. We have primarily considered relational properties defined over multiple executions of the same DNN, however, RaVeN can be extended to relational properties involving two or more different DNNs - local equivalence of pair of DNNs [58], properties defined over an ensemble of DNNs, etc. RaVeN can also be integrated inside the training loop to obtain more trustworthy and safe neural networks. We leave this as future work. Also, the current implementation of RaVeN is sequential but as stated above certain steps like the product DNN analysis and pairwise difference computation with DiffPoly can be parallelized to reduce the verification cost.

ACKNOWLEDGEMENT

We thank the anonymous reviewers for their insightful comments. This work was supported in part by NSF Grants No. CCF-2238079, CCF-2316233, CNS-2148583, Google Research Scholar award, and Qualcomm Innovation Fellowship.

ARTIFACT STATEMENT

The artifact, on which the evaluation was done, is available at: https://zenodo.org/records/10807316 with DOI 10.5281/zenodo.10807316 [23]. The artifact includes instructions to reproduce the claimed results of the paper.

REFERENCES

[1] Aws Albarghouthi. 2021. Introduction to Neural Network Verification. *Found. Trends Program. Lang.* 7, 1-2 (2021), 1–157. https://doi.org/10.1561/2500000051

[2] Filippo Amato, Alberto López, Eladia María Peña-Méndez, Petr Vaňhara, Aleš Hampl, and Josef Havel. 2013. Artificial neural networks in medical diagnosis. *Journal of Applied Biomedicine* 11, 2 (2013).

[3] Greg Anderson, Shankara Pailoor, Isil Dillig, and Swarat Chaudhuri. 2019. Optimization and Abstraction: A Synergistic Approach for Analyzing Neural Network Robustness. In *Proc. Programming Language Design and Implementation (PLDI)*. 731–744.

[4] Stanley Bak, Taylor Dohmen, K. Subramani, Ashutosh Trivedi, Alvaro Velasquez, and Piotr Wojciechowski. 2023. The Octatope Abstract Domain for Verification of Neural Networks. In *Formal Methods - 25th International Symposium, FM 2023, Lübeck, Germany, March 6-10, 2023, Proceedings (Lecture Notes in Computer Science, Vol. 14000)*, Marsha Chechik, Joost-Pieter Katoen, and Martin Leucker (Eds.). Springer, 454–472. https://doi.org/10.1007/978-3-031-27481-7_26

[5] Stanley Bak, Hoang-Dung Tran, Kerianne Hobbs, and Taylor T. Johnson. 2020. Improved Geometric Path Enumeration for Verifying ReLU Neural Networks. In *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12224)*, Shuvendu K. Lahiri and Chao Wang (Eds.). Springer, 66–96. https://doi.org/10.1007/978-3-030-53288-8_4

[6] Mislav Balunovic, Maximilian Baader, Gagandeep Singh, Timon Gehr, and Martin Vechev. 2019. Certifying Geometric Robustness of Neural Networks. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/f7fa6aca028e7ff4ef62d75ed025fe76-Paper.pdf

[7] G. Barthe, P.R. D'Argenio, and T. Rezk. 2004. Secure information flow by self-composition. In *Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004.* 100–114. https://doi.org/10.1109/CSFW.2004.1310735

[8] Barry Becker and Ronny Kohavi. 1996. Adult. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5XW20.

[9] Raven Beutner and Bernd Finkbeiner. 2022. Software Verification of Hyperproperties Beyond k-Safety. In *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13371)*, Sharon Shoham and Yakir Vizel (Eds.). Springer, 341–362.

[10] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[11] Laura Bozzelli, Adriano Peron, and César Sánchez. 2021. Asynchronous Extensions of HyperLTL. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021.* IEEE, 1–13. https://doi.org/10.1109/LICS52264.2021.9470583

[12] Christopher Brix, Mark Niklas Müller, Stanley Bak, Taylor T Johnson, and Changliu Liu. 2023. First three years of the international verification of neural networks competition (VNN-COMP). *International Journal on Software Tools for Technology Transfer* (2023), 1–11.

[13] Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Pushmeet Kohli, P Torr, and P Mudigonda. 2020. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research* 21, 2020 (2020).

[14] Rudy R Bunel, Oliver Hinder, Srinadh Bhojanapalli, and Krishnamurthy Dvijotham. 2020. An efficient nonconvex reformulation of stagewise convex optimization problems. *Advances in Neural Information Processing Systems* 33 (2020).

[15] Jacob Burnim and Koushik Sen. 2009. Asserting and Checking Determinism for Multithreaded Programs. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering* (Amsterdam, The Netherlands) *(ESEC/FSE '09)*. Association for Computing Machinery, New York, NY, USA, 3–12. https://doi.org/10.1145/1595696.1595700

[16] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*. Ieee, 39–57.

[17] Maria Christakis, Hasan Ferit Eniser, Jörg Hoffmann, Adish Singla, and Valentin Wüstholz. 2022. Specifying and Testing k-Safety Properties for Machine-Learning Models. *arXiv preprint arXiv:2206.06054* (2022).

[18] Berkeley R. Churchill, Oded Padon, Rahul Sharma, and Alex Aiken. 2019. Semantic program alignment for equivalence checking. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*, Kathryn S. McKinley and Kathleen Fisher (Eds.). ACM, 1027–1040. https://doi.org/10.1145/3314221.3314596

[19] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. 2019. Certified Adversarial Robustness via Randomized Smoothing. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 1310–1320. https://proceedings.mlr.press/v97/cohen19c.html

[20] Patrick Cousot and Nicolas Halbwachs. 1978. Automatic Discovery of Linear Restraints among Variables of a Program. In *Proceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (Tucson, Arizona) *(POPL '78)*. Association for Computing Machinery, New York, NY, USA, 84–96. https://doi.org/10.1145/512760.512770

[21] Hennie Daniels and Marina Velikova. 2010. Monotone and partially monotone neural networks. *IEEE Transactions on Neural Networks* 21, 6 (2010), 906–917.

[22] Sumanth Dathathri, Krishnamurthy Dvijotham, Alexey Kurakin, Aditi Raghunathan, Jonathan Uesato, Rudy Bunel, Shreya Shankar, Jacob Steinhardt, Ian J. Goodfellow, Percy Liang, and Pushmeet Kohli. 2020. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/397d6b4c83c91021fe928a8c4220386b-Abstract.html

[23] D.Banerjee. 2024. RaVeN: v1.1. https://doi.org/10.5281/zenodo.10807316

[24] Hai Duong, Linhan Li, ThanhVu Nguyen, and Matthew Dwyer. 2023. A DPLL (T) Framework for Verifying Deep Neural Networks. *arXiv preprint arXiv:2307.10266* (2023).

[25] Ruediger Ehlers. 2017. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*.

[26] Marco Eilers, Peter Müller, and Samuel Hitz. 2018. Modular Product Programs. In *Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10801)*, Amal Ahmed (Ed.). Springer, 502–529. https://doi.org/10.1007/978-3-319-89884-1_18

[27] Azadeh Farzan and Anthony Vandikas. 2019. Automated Hypersafety Verification. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11561)*, Isil Dillig and Serdar Tasiran (Eds.). Springer, 200–218. https://doi.org/10.1007/978-3-030-25540-4_11

[28] Claudio Ferrari, Mark Niklas Mueller, Nikola Jovanović, and Martin Vechev. 2022. Complete Verification via Multi-Neuron Relaxation Guided Branch-and-Bound. In *International Conference on Learning Representations*. https://openreview.net/forum?id=l_amHf1oaK

[29] Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. 2015. Algorithms for Model Checking HyperLTL and HyperCTL ˆ*. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9206)*, Daniel Kroening and Corina S. Pasareanu (Eds.). Springer, 30–48. https://doi.org/10.1007/978-3-319-21690-4_3

[30] Aymeric Fromherz, Klas Leino, Matt Fredrikson, Bryan Parno, and Corina Pasareanu. 2021. Fast Geometric Projections for Local Robustness Certification. In *International Conference on Learning Representations*. https://openreview.net/forum?id=zWy1uxjDdZJ

[31] 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)*.

[32] Chuqin Geng, Nham Le, Xiaojie Xu, Zhaoyue Wang, Arie Gurfinkel, and Xujie Si. 2023. Towards reliable neural specifications. In *International Conference on Machine Learning*. PMLR, 11196–11212.

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

[34] Akhil Gupta, Naman Shukla, Lavanya Marla, Arinbjörn Kolbeinsson, and Kartik Yellepeddi. 2019. How to incorporate monotonicity in deep networks while preserving flexibility? *arXiv preprint arXiv:1909.10662* (2019).

[35] Gurobi Optimization, LLC. 2018. Gurobi Optimizer Reference Manual.

[36] Richard W Hamming. 1950. Error detecting and error correcting codes. *The Bell system technical journal* 29, 2 (1950), 147–160.

[37] David Harrison Jr and Daniel L Rubinfeld. 1978. Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management* 5, 1 (1978), 81–102.

[38] Anan Kabaha and Dana Drachsler-Cohen. 2022. Boosting Robustness Verification of Semantic Feature Neighborhoods. In *Static Analysis - 29th International Symposium, SAS 2022, Auckland, New Zealand, December 5-7, 2022, Proceedings (Lecture Notes in Computer Science, Vol. 13790)*, Gagandeep Singh and Caterina Urban (Eds.). Springer, 299–324. https://doi.org/10.1007/978-3-031-22308-2_14

[39] Guy Katz, Derek Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David Dill, Mykel Kochenderfer, and Clark Barrett. 2019. *The Marabou Framework for Verification and Analysis of Deep Neural Networks.* 443–452.

[40] Haitham Khedr and Yasser Shoukry. 2023. CertiFair: A Framework for Certified Global Fairness of Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence* 37, 7 (Jun. 2023), 8237–8245.

[41] Máté Kovács, Helmut Seidl, and Bernd Finkbeiner. 2013. Relational abstract interpretation for the verification of 2-hypersafety properties. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung (Eds.). ACM, 211–222. https://doi.org/10.1145/2508859.2516721

[42] Alex Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images. (2009).

[43] Jianglin Lan, Yang Zheng, and Alessio Lomuscio. 2022. Tight Neural Network Verification via Semidefinite Relaxations and Linear Reformulations. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022.* AAAI Press, 7272–7280. https://ojs.aaai.org/index.php/AAAI/article/view/20689

[44] Jacob Laurel, Siyuan Brant Qian, Gagandeep Singh, and Sasa Misailovic. 2023. Synthesizing precise static analyzers for automatic differentiation. *Proceedings of the ACM on Programming Languages* 7, OOPSLA2 (2023), 1964–1992.

[45] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. 1989. Handwritten Digit Recognition with a Back-Propagation Network. In *NIPS.* 396–404.

[46] Juncheng Li, Shuhui Qu, Xinjian Li, Joseph Szurley, J. Zico Kolter, and Florian Metze. 2019. Adversarial Music: Real world Audio Adversary against Wake-word Detection System. In *Proc. Neural Information Processing Systems (NeurIPS).* 11908–11918.

[47] Juncheng Li, Frank R. Schmidt, and J. Zico Kolter. 2019. Adversarial camera stickers: A physical camera-based attack on deep learning systems. In *Proc. International Conference on Machine Learning, ICML*, Vol. 97. 3896–3904.

[48] Xingchao Liu, Xing Han, Na Zhang, and Qiang Liu. 2020. Certified monotonic neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 15427–15438.

[49] Zikun Liu, Changming Xu, Emerson Sie, Gagandeep Singh, and Deepak Vasisht. 2023. Exploring Practical Vulnerabilities of Machine Learning-based Wireless Systems. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023.* USENIX Association, 1801–1817.

[50] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations.* https://openreview.net/forum?id=rJzIBfZAb

[51] Antoine Miné. 2001. A new numerical abstract domain based on difference-bound matrices. In *Programs as Data Objects: Second Symposium, PADO2001 Aarhus, Denmark, May 21–23, 2001 Proceedings.* Springer, 155–172.

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

[53] Matthew Mirman, Timon Gehr, and Martin Vechev. 2018. Differentiable Abstract Interpretation for Provably Robust Neural Networks. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, 3578–3586. https://proceedings.mlr.press/v80/mirman18b.html

[54] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. 2017. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 1765–1773.

[55] Mark Niklas Müller, Franziska Eckert, Marc Fischer, and Martin T. Vechev. 2023. Certified Training: Small Boxes are All You Need. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023.* OpenReview.net. https://openreview.net/pdf?id=7oFuxtJtUMH

[56] Satoshi Munakata, Caterina Urban, Haruki Yokoyama, Koji Yamamoto, and Kazuki Munakata. 2023. Verifying Attention Robustness of Deep Neural Networks Against Semantic Perturbations. In *NASA Formal Methods - 15th International Symposium, NFM 2023, Houston, TX, USA, May 16-18, 2023, Proceedings (Lecture Notes in Computer Science, Vol. 13903)*, Kristin Yvonne Rozier and Swarat Chaudhuri (Eds.). Springer, 37–61. https://doi.org/10.1007/978-3-031-33170-1_3

[57] Alessandro De Palma, Harkirat S. Behl, Rudy Bunel, Philip H. S. Torr, and M. Pawan Kumar. 2021. Scaling the Convex Barrier with Active Sets. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=uQfOy7LrlTR

[58] Brandon Paulsen, Jingbo Wang, and Chao Wang. 2020. ReluDiff: Differential Verification of Deep Neural Networks. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 714–726. https://doi.org/10.1145/3377811.3380337

[59] Brandon Paulsen, Jingbo Wang, Jiawei Wang, and Chao Wang. 2021. NeuroDiff: Scalable Differential Verification of Neural Networks Using Fine-Grained Approximation. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering* (Virtual Event, Australia) *(ASE '20)*. Association for Computing Machinery, New York, NY, USA, 784–796. https://doi.org/10.1145/3324884.3416560

[60] Yannik Potdevin, Dirk Nowotka, and Vijay Ganesh. 2019. An empirical investigation of randomized defenses against adversarial attacks. *arXiv preprint arXiv:1909.05580* (2019).

[61] Rob Potharst and Adrianus Johannes Feelders. 2002. Classification trees for problems with monotonicity constraints. *ACM SIGKDD Explorations Newsletter* 4, 1 (2002), 1–10.

[62] Chongli Qin, Krishnamurthy (Dj) Dvijotham, Brendan O'Donoghue, Rudy Bunel, Robert Stanforth, Sven Gowal, Jonathan Uesato, Grzegorz Swirszcz, and Pushmeet Kohli. 2019. Verification of Non-Linear Specifications for Neural Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HyeFAsRctQ

[63] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. 2019. A Convex Relaxation Barrier to Tight Robustness Verification of Neural Networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada.*

[64] Joseph Scott, Guanting Pan, Elias B. Khalil, and Vijay Ganesh. 2022. Goose: A Meta-Solver for Deep Neural Network Verification. In *Proceedings of the 20th Internal Workshop on Satisfiability Modulo Theories co-located with the 11th International Joint Conference on Automated Reasoning (IJCAR 2022) part of the 8th Federated Logic Conference (FLoC 2022), Haifa, Israel, August 11-12, 2022 (CEUR Workshop Proceedings, Vol. 3185)*, David Déharbe and Antti E. J. Hyvärinen (Eds.). CEUR-WS.org, 99–113. https://ceur-ws.org/Vol-3185/extended678.pdf

[65] Ron Shemer, Arie Gurfinkel, Sharon Shoham, and Yakir Vizel. 2019. Property Directed Self Composition. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11561)*, Isil Dillig and Serdar Tasiran (Eds.). Springer, 161–179. https://doi.org/10.1007/978-3-030-25540-4_9

[66] Zhouxing Shi, Yihan Wang, Huan Zhang, J Zico Kolter, and Cho-Jui Hsieh. 2022. Efficiently computing local lipschitz constants of neural networks via bound propagation. *Advances in Neural Information Processing Systems* 35 (2022), 2350–2364.

[67] Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. 2019. Beyond the single neuron convex barrier for neural network certification. In *Advances in Neural Information Processing Systems*.

[68] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. 2018. Fast and effective robustness certification. *Advances in Neural Information Processing Systems* 31 (2018).

[69] 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).

[70] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. Robustness Certification with Refinement. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HJgeEh09KQ

[71] Matthew Sotoudeh, Zhe Tao, and Aditya V Thakur. 2023. SyReNN: a tool for analyzing deep neural networks. *International Journal on Software Tools for Technology Transfer* 25, 2 (2023), 145–165.

[72] Matthew Sotoudeh and Aditya V Thakur. 2020. Abstract neural networks. In *Static Analysis: 27th International Symposium, SAS 2020, Virtual Event, November 18–20, 2020, Proceedings 27*. Springer, 65–88.

[73] Marcelo Sousa and Isil Dillig. 2016. Cartesian hoare logic for verifying k-safety properties. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*, Chandra Krintz and Emery D. Berger (Eds.). ACM, 57–69. https://doi.org/10.1145/2908080.2908092

[74] Cheng Tan, Yibo Zhu, and Chuanxiong Guo. 2021. Building Verified Neural Networks with Specifications for Systems. In *Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems* (Hong Kong, China) *(APSys '21)*. 42–47.

[75] Hoang-Dung Tran, Diago Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T. Johnson. 2019. Star-Based Reachability Analysis of Deep Neural Networks. In *Formal Methods – The Next 30 Years*, Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira (Eds.). Springer International Publishing, Cham, 670–686.

[76] Shubham Ugare, Gagandeep Singh, and Sasa Misailovic. 2022. Proof transfer for fast certification of multiple approximate neural networks. *Proc. ACM Program. Lang.* 6, OOPSLA1 (2022), 1–29. https://doi.org/10.1145/3527319

[77] Hiroshi Unno, Tachio Terauchi, and Eric Koskinen. 2021. Constraint-Based Relational Verification. In *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I (Lecture*

Notes in Computer Science, Vol. 12759), Alexandra Silva and K. Rustan M. Leino (Eds.). Springer, 742–766. https://doi.org/10.1007/978-3-030-81685-8_35

[78] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*.

[79] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 2021. Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Complete and Incomplete Neural Network Verification. *arXiv preprint arXiv:2103.06624* (2021).

[80] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 2021. Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Neural Network Robustness Verification. In *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (Eds.). https://openreview.net/forum?id=ahYIlRBeCFw

[81] Zhilu Wang, Chao Huang, and Qi Zhu. 2022. Efficient global robustness certification of neural networks via interleaving twin-network encoding. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1087–1092.

[82] Eric Wong and J. Zico Kolter. 2018. Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018 (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 5283–5292. http://proceedings.mlr.press/v80/wong18a.html

[83] Haoze Wu, Clark Barrett, Mahmood Sharif, Nina Narodytska, and Gagandeep Singh. 2022. Scalable Verification of GNN-Based Job Schedulers. *Proc. ACM Program. Lang.* 6, OOPSLA2, Article 162 (oct 2022), 30 pages. https://doi.org/10.1145/3563325

[84] Haoze Wu, Teruhiro Tagomori, Alexander Robey, Fengjun Yang, Nikolai Matni, George Pappas, Hamed Hassani, Corina Pasareanu, and Clark Barrett. 2023. Toward certified robustness against real-world distribution shifts. In *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*. IEEE, 537–553.

[85] Changming Xu and Gagandeep Singh. 2022. Robust Universal Adversarial Perturbations. *CoRR* abs/2206.10858 (2022). https://doi.org/10.48550/arXiv.2206.10858 arXiv:2206.10858

[86] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. 2020. Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond. (2020).

[87] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. 2021. Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers. In *International Conference on Learning Representations*. https://openreview.net/forum?id=nVZtXBI6LNn

[88] Yi Zeng, Zhouxing Shi, Ming Jin, Feiyang Kang, Lingjuan Lyu, Cho-Jui Hsieh, and Ruoxi Jia. 2023. Towards Robustness Certification Against Universal Perturbations. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=7GEvPKxjtt

[89] Mustafa Zeqiri, Mark Niklas Müller, Marc Fischer, and Martin T. Vechev. 2023. Efficient Certified Training and Robustness Verification of Neural ODEs. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/pdf?id=KyoVpYvWWnK

[90] Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh. 2020. Towards stable and efficient training of verifiably robust neural networks. In *Proc. International Conference on Learning Representations (ICLR)*.

[91] Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 2022. General Cutting Planes for Bound-Propagation-Based Neural Network Verification. In *Advances in Neural Information Processing Systems*, Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (Eds.). https://openreview.net/forum?id=5haAJAcofjc

[92] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. 2018. Efficient neural network robustness certification with general activation functions. *Advances in neural information processing systems* 31 (2018).

Received 2023-11-16; accepted 2024-03-31

8 ADDITIONAL BACKGROUND

8.1 Adversarial Perturbations

An *adversarial perturbation*, v, added to an input, x, it is attacking is an *adversarial example*, $x' = x + v$. Additionally, x' is only adversarial if it causes the target model to misclassify, in other words, if $f(x) = y$ then $f(x') \neq y$. It is typically assumed that these perturbations are small so as they do not effect the semantic context of the image (a human would still correctly classify the adversarial example). The most common bound is an L_p bound, i.e $\|X\|_p \leq \epsilon$.

In the case where standard adversarial perturbations are not feasible, verification against universal adversarial perturbations (UAPs) is desirable. A UAP consists of a single perturbation u which is adversarial for many inputs. We will start by formally defining strong UAPs.

8.2 Universal Adversarial Perturbations

An *universal adversarial perturbation* (UAP), u, added to an input, x, causes the target model to misclassify on a set of inputs $X_1, \ldots X_k$, in other words, if $\forall i \in [k].f(X_i) = y_i$ then $\forall i \in [k].f(X_i + u) \neq y_i$. Formally,

Definition 8.1. A *universal adversarial perturbation* is a vector $\mathbf{u} \in \mathbb{R}^d$ which, when added to all datapoints in μ causes the classifier f to misclassify. Formally, given γ, a bound on universal ASR, and l_p-norm with corresponding bound ϵ, $\mathbf{u}$ is a UAP iff $\forall x, y \in \mu f(x) \neq y$ and $||\mathbf{u}||_p < \epsilon$.

8.3 UAP verification

Definition 8.2 (UAP Verification Problem). Given points $X^* = X_1^*, ..., X_k^* \in \mathbb{R}^{n_0}$ and $\epsilon \in \mathbb{R}$ we can first define individual input constraints $\forall i \in [k].\phi_{in}^i = \|X_i^* - X_i\|_\infty \leq \epsilon$. We define Φ^δ as follows:

$$\Phi^\delta(X_1, \ldots, X_k) = \bigwedge_{(i,j \in [k]) \wedge (i<j)} (X_i - X_j = X_i^* - X_j^*) \tag{11}$$

Then, we have $\Phi = \bigwedge_{i=1}^k \phi_{in}^i \wedge \Phi^\delta$. Next, we define Ψ as conjunction of $k \times n_l$ clauses where $\forall a \in [k], \forall b \in [n_l]$ the clause $\psi_{a,b}$ is defined as $\psi_{a,b} = (C_{a,b}^T Y_a \geq 0)$ and $C_{a,b} \in \mathbb{R}^{n_l}$ is given below

$$\forall i \in [n_l].c_{a,b,i} = \begin{cases} 1 & \text{if } i \neq b \text{ and } i \text{ is the correct label for } Y_a \\ -1 & \text{if } i = b \text{ and } i \text{ is not the correct label for } Y_a \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

8.4 Targeted UAP verification

Unlike the unrestricted UAP attack above, in targeted UAP, the attacker tries to make the DNN misclassify inputs to a given class. Here we check whether all inputs can be classified as a target class t by adding the same perturbation to each input. The formal definition of the targeted UAP verification problem is in .

Definition 8.3 (Targeted UAP Verification Problem). Given points $X^* = X_1^*, \ldots, X_k^* \in \mathbb{R}^{n_0}$, $\epsilon \in \mathbb{R}$, and target label t, the targeted UAP verification problem has the same input specification as the UAP verification problem, seen in Definition 8.2. Next, we define Ψ as conjunction of $k \times n_l$ clauses where $\forall a \in [k], \forall b \in [n_l]$ the clause $\psi_{a,b}$ is defined as $\psi_{a,b} = (C_{a,b}^T Y_a \geq 0)$ and $C_{a,b} \in \mathbb{R}^{n_l}$ is:

$$\forall i \in [n_l].c_{a,b,i} = \begin{cases} 1 & \text{if } i \neq b \text{ and } i = t \\ -1 & \text{if } i = b \text{ and } i \neq t \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

8.5 Worst case Hamming distance verification

Definition 8.4. Given points $X^* = X_1^*, ..., X_k^* \in \mathbb{R}^{n_0}$, $\epsilon \in \mathbb{R}$, and a binary digit classifier neural network $N_2 : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^2$ we can define a binary digit string $s \in \{0, 1\}^k$ as the conjunction of the output of N_2 on each input $\forall i \in [k].X_i$ where each X_i is an image of a binary digit. We are interested in bounding the worst-case hamming distance between s^*, the binary digit string classified by N_2, and s the actual binary digit string corresponding to list of perturbed images $\forall i \in [k].X_i + V$ s.t. $V \in \mathbb{R}^{n_0}$ and $|V|_\infty \le \epsilon$. Given these definitions, we can use the Φ and Ψ defined in Definition 8.2.

8.6 Monotonicity verification

Definition 8.5 (Monotonic Verification Problem). Given a point $X^* \in \mathbb{R}^{n_0}$, $\epsilon \in \mathbb{R}$, network $N_m : \mathbb{R}^{n_0} \to \mathbb{R}$, monotonic input dimension $m \in [n_0]$, monotonic direction $d \in \{-1, 1\}$, let $C_j \in \mathbb{R}^{n_0}$ be the one-hot vector defined as all 0's except for a 1 in the j^{th} dimension and $j \in [n_0]$. We can define $\forall i \in [2].\phi^i_{in} = (\|C^T_m X^* - C^T_m X_i\|_\infty \le \epsilon) \wedge \varphi_i$ where $\varphi_i = \wedge_{j \in [n_0] \wedge (j \neq m)} (\|C^T_j X^* - C^T_j X_i\|_\infty = 0)$. Now, we can define $\phi^\delta = C^T_m X_1 - C^T_m X_2 > 0$ and $\Phi = \phi^1_{in} \wedge \phi^2_{in} \wedge \phi^\delta$. Finally, our output specification can be defined as $\Psi(N_m(X_1), N_m(X_2)) = d \cdot (N_m(X_1) - N_m(X_2)) \ge 0$.

8.7 Detailed execution of DeepZ abstract transformer on the example Product DNN

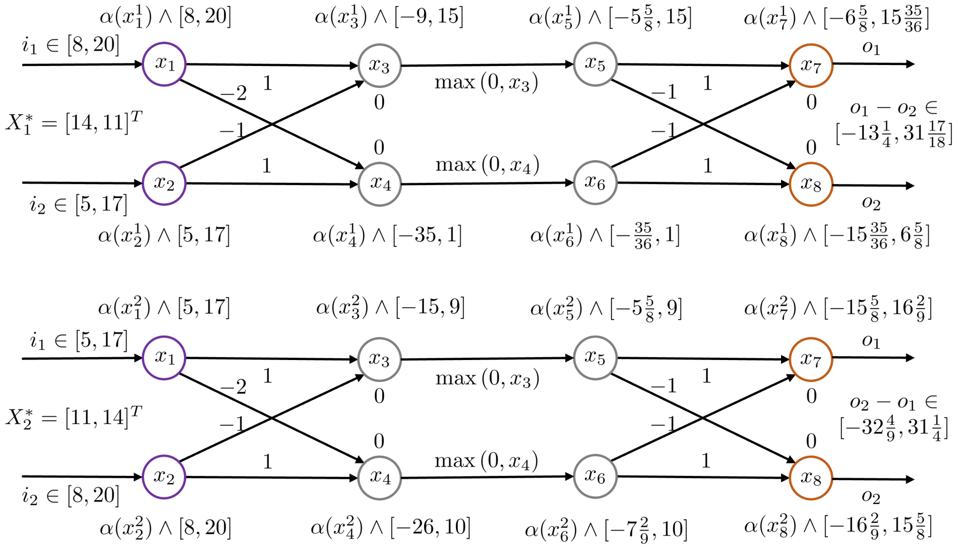

$\alpha(x^1_1) \wedge [8, 20]$
$\alpha(x^1_3) \wedge [-9, 15]$
$\alpha(x^1_5) \wedge [-5\frac{5}{8}, 15]$
$\alpha(x^1_7) \wedge [-6\frac{5}{8}, 15\frac{35}{36}]$
$i_1 \in [8, 20]$
x_1
x_3
x_5
x_7
o_1
-2
1
$\max(0, x_3)$
-1
1
0
$X^*_1 = [14, 11]^T$
-1
-1
$o_1 - o_2 \in [-13\frac{1}{4}, 31\frac{17}{18}]$
0
0
1
$\max(0, x_4)$
1
x_2
x_4
x_6
x_8
$i_2 \in [5, 17]$
o_2
$\alpha(x^1_2) \wedge [5, 17]$
$\alpha(x^1_4) \wedge [-35, 1]$
$\alpha(x^1_6) \wedge [-\frac{35}{36}, 1]$
$\alpha(x^1_8) \wedge [-15\frac{35}{36}, 6\frac{5}{8}]$
$\alpha(x^2_1) \wedge [5, 17]$
$\alpha(x^2_3) \wedge [-15, 9]$
$\alpha(x^2_5) \wedge [-5\frac{5}{8}, 9]$
$\alpha(x^2_7) \wedge [-15\frac{5}{8}, 16\frac{2}{9}]$
$i_1 \in [5, 17]$
x_1
x_3
x_5
x_7
o_1
-2
1
$\max(0, x_3)$
-1
1
0
$X^*_2 = [11, 14]^T$
-1
-1
$o_2 - o_1 \in [-32\frac{4}{9}, 31\frac{1}{4}]$
0
0
1
$\max(0, x_4)$
1
x_2
x_4
x_6
x_8
$i_2 \in [8, 20]$
o_2
$\alpha(x^2_2) \wedge [8, 20]$
$\alpha(x^2_4) \wedge [-26, 10]$
$\alpha(x^2_6) \wedge [-7\frac{2}{9}, 10]$
$\alpha(x^2_8) \wedge [-16\frac{2}{9}, 15\frac{5}{8}]$

Fig. 12. Product DNN analysis on input regions ϕ^1_t and ϕ^2_t using DeepZ

First, we compute the zonotope expression, concrete lower bound, and concrete upper bound of the input variables of both $N^{X_1}_{ex}$ and $N^{X_2}_{ex}$. Note, the concrete lower bound, and concrete upper bound of any variable are obtained by calculating the minimum and maximum value of the zonotope expression associated with that variable.

$$\alpha(i^1_1) = 14 + 6 \cdot \eta^1_1 \qquad \alpha(i^1_2) = 11 + 6 \cdot \eta^1_2 \qquad \alpha(x^1_1) = 14 + 6 \cdot \eta^1_1 \qquad \alpha(x^1_2) = 11 + 6 \cdot \eta^1_2$$
$$\alpha(i^2_1) = 11 + 6 \cdot \eta^2_1 \qquad \alpha(i^2_2) = 14 + 6 \cdot \eta^2_2 \qquad \alpha(x^2_1) = 11 + 6 \cdot \eta^2_1 \qquad \alpha(x^2_2) = 14 + 6 \cdot \eta^2_2$$
$$x^1_1 \in [8, 20] \qquad x^1_2 \in [5, 17] \qquad x^2_1 \in [5, 17] \qquad x^2_2 \in [8, 20]$$

Next, the affine transform at the first layer computes the zonotope expressions for variables x^1_3, x^1_4, x^2_3, and x^2_4 as shown below.

$$\alpha(x^1_3) = (14 + 6 \cdot \eta^1_1) - (11 + 6 \cdot \eta^1_2) = 3 + 6 \cdot \eta^1_1 - 6 \cdot \eta^1_2 \qquad \alpha(x^1_4) = -17 - 12 \cdot \eta^1_1 + 6 \cdot \eta^1_2$$
$$\alpha(x^2_3) = (11 + 6\eta^2_1) - (14 + 6 \cdot \eta^2_2) = -3 + 6 \cdot \eta^2_1 - 6 \cdot \eta^2_2 \qquad \alpha(x^2_4) = -8 - 12 \cdot \eta^2_1 + 6 \cdot \eta^2_2$$

Next, we use the ReLU transformer proposed in [68] to compute the zonotope expression associate with the variables x_5^1, x_6^1, x_5^2, and x_6^2 from the zonotope expression of x_3^1, x_4^1, x_3^2, and x_4^2. First, we describe the ReLU transformer ($ReLU^{\sharp}$) below where for any zonotope expression $\alpha(x) = v + \sum_{i=1}^{n} w_i \cdot \eta_i$ ($v \in \mathbb{R}$ and $w \in \mathbb{R}^n$) for any real $\lambda \in \mathbb{R}, \mu \in \mathbb{R}$ the zonotope expression $\lambda \cdot \alpha(x) + \mu$ denotes $\lambda \cdot \alpha(x) + \mu = \lambda \cdot v + \sum_{i=1}^{n}(\lambda \cdot w_i) \cdot \eta_i$, l_x and u_x denotes the concrete lower bound and concrete upper bound of the variable x respectively and η_{new} denotes a new noise variable.

$$ReLU^{\sharp}(\alpha(x)) = \begin{cases} \alpha(x) \quad \text{if } l_x \geq 0 \\ 0 \quad \text{if } u_x \leq 0 \\ \lambda \cdot \alpha(x) + \mu + \mu \cdot \eta_{new} \quad \text{if } (l_x < 0) \wedge (u_x > 0) \text{ where } \lambda = \frac{u_x}{u_x - l_x} \text{ and } \mu = -\frac{u_x \cdot l_x}{2 \cdot (u_x - l_x)} \end{cases}$$

For soundness proof of $ReLU^{\sharp}$ refer to Theorem 3.1 of [68]. Using the the $ReLU$ transformer $ReLU^{\sharp}$ we can compute the zonotope expression associated with x_5^1, x_6^1, x_5^2, and x_6^2. For example, we show the computation of the zonotope expression $\alpha(x_5^1)$ below.

$$\alpha(x_5^1) = \lambda \cdot \alpha(x_3^1) + \mu + \mu \cdot \eta_5^1 \quad \text{where } \lambda = \frac{u_{x_3^1}}{u_{x_3^1} - l_{x_3^1}} \text{ and } \mu = -\frac{u_{x_3^1} \cdot l_{x_3^1}}{2 \cdot (u_{x_3^1} - l_{x_3^1})}$$

For the variables in the final layer x_7^1, x_8^1, x_7^2, and x_8^2 and subsequently for the output variables o_1^1, o_2^1, o_1^2, and o_2^2 we compute the zonotope expressions by applying the affine transform on the zonotope expressions associated with the variables x_5^1, x_6^1, x_5^2, and x_6^2. For example, we show the computation of the zonotope expression $\alpha(x_7^1)$ below.

$$\alpha(o_1^1) = \alpha(x_7^1) = \alpha(x_5^1) - \alpha(x_5^1) = 9.347 + 8.167\eta_1^1 - 7.833\eta_2^1 + 5.625\eta_3^1 - 0.972\eta_4^1$$

8.8 Detailed DiffPoly constraints on x_i^1 & x_i^2 for the illustrative example

$x_1^{1,\leq} = 8$	$x_1^{1,\geq} = 20$	$l_{1,x_1} = 8$	$u_{1,x_1} = 20$
$x_2^{1,\leq} = 5$	$x_2^{1,\geq} = 17$	$l_{1,x_2} = 5$	$u_{1,x_2} = 17$
$x_3^{1,\leq} = x_1^1 - x_2^1$	$x_3^{1,\geq} = x_1^1 - x_2^1$	$l_{1,x_3} = -9$	$u_{1,x_3} = 15$
$x_4^{1,\leq} = -2 \cdot x_1^1 + x_2^1$	$x_4^{1,\geq} = -2 \cdot x_1^1 + x_2^1$	$l_{1,x_4} = -35$	$u_{1,x_4} = 1$
$x_5^{1,\leq} = x_3^1$	$x_5^{1,\geq} = \frac{5}{24} \cdot x_3^1 + \frac{45}{8}$	$l_{1,x_5} = -5\frac{5}{8}$	$u_{1,x_5} = 15$
$x_6^{1,\leq} = 0$	$x_6^{1,\geq} = \frac{1}{36} \cdot x_4^1 + \frac{35}{36}$	$l_{1,x_6} = -\frac{35}{36}$	$u_{1,x_6} = 1$
$x_7^{1,\leq} = x_5^1 - x_6^1$	$x_7^{1,\geq} = x_5^1 - x_6^1$	$l_{1,x_7} = -6\frac{5}{8}$	$u_{1,x_7} = 15\frac{35}{36}$
$x_8^{1,\leq} = -x_5^1 + x_6^1$	$x_8^{1,\geq} = -x_5^1 + x_6^1$	$l_{1,x_8} = -15\frac{35}{36}$	$u_{1,x_8} = 16\frac{2}{9}$

$$
\begin{array}{llll}
x_1^{2,\leq} = 5 & x_1^{2,\geq} = 17 & l_{2,x_1} = 5 & u_{2,x_1} = 17 \\
x_2^{2,\leq} = 8 & x_2^{2,\geq} = 20 & l_{2,x_2} = 8 & u_{2,x_2} = 20 \\
x_3^{2,\leq} = x_1^2 - x_2^2 & x_3^{2,\geq} = x_1^2 - x_2^2 & l_{2,x_3} = -15 & u_{2,x_3} = 9 \\
x_4^{2,\leq} = -2 \cdot x_1^2 + x_2^2 & x_4^{2,\geq} = -2 \cdot x_1^2 + x_2^2 & l_{2,x_4} = -26 & u_{2,x_4} = 10 \\
x_5^{2,\leq} = 0 & x_5^{2,\geq} = \frac{3}{8} \cdot x_3^2 + \frac{45}{8} & l_{2,x_5} = -5\frac{5}{8} & u_{2,x_5} = 9 \\
x_6^{2,\leq} = 0 & x_6^{2,\geq} = \frac{5}{18} \cdot x_4^2 + \frac{65}{9} & l_{2,x_6} = -7\frac{2}{9} & u_{2,x_6} = 10 \\
x_7^{2,\leq} = x_5^2 - x_6^2 & x_7^{2,\geq} = x_5^2 - x_6^2 & l_{2,x_7} = -15\frac{5}{8} & u_{2,x_7} = 16\frac{2}{9} \\
x_8^{2,\leq} = -x_5^2 + x_6^2 & x_8^{2,\geq} = -x_5^2 + x_6^2 & l_{2,x_8} = -16\frac{2}{9} & u_{2,x_8} = 15\frac{5}{8}
\end{array}
$$

9 MILPS FOR THE ILLUSTRATIVE EXAMPLE

9.1 MILP formulation from state-of-the-art baseline [88]

The state-of-the-art baseline relates output variables as linear constraints over the input variables based on the analysis of an existing non-relational verifier (in this case DeepZ) on the product DNN. The cross-execution constraints (shown in blue) are only tracked at the input layer. The optimal value of t and the verification result for this formulation is shown below.

$$
\begin{aligned}
\text{min} \;\; & t \\
& \text{subject to} \\
& min(F_1) = z_1, z_1 \leq t, min(F_2) = z_2, z_2 \leq t \quad \text{[MILP encoding of } \Psi\text{]} \\
& F_1 = o_1^1 - o_2^1, F_2 = -o_1^2 + o_2^2 \\
& x_1^1 = 14 + 6 * \eta_1^1, x_2^1 = 11 + 6 * \eta_2^1 \\
& x_1^2 = 11 + 6 * \eta_1^2, x_2^2 = 14 + 6 * \eta_2^2 \\
& (x_1^1 - x_1^2) = 3, (x_2^1 - x_2^2) = -3 \quad \text{[cross-execution constraints at input layer]} \\
& o_1^1 = 9.347 + 8.167\eta_1^1 - 7.833\eta_2^1 + 5.625\eta_3^1 - 0.972\eta_4^1 \\
& o_2^1 = -9.347 - 8.167\eta_1^1 + 7.833\eta_2^1 - 5.625\eta_3^1 + 0.972\eta_4^1 \\
& o_1^2 = -0.597 - 11.167\eta_1^2 + 7.833\eta_2^2 - 5.625\eta_3^2 + 7.222\eta_4^2 \\
& o_2^2 = 0.597 + 11.167\eta_1^2 - 7.833\eta_2^2 + 5.625\eta_3^2 - 7.222\eta_4^2 \\
& -1 \leq \eta_i^j \leq 1 \;\; \forall i \in \{1,2,3,4\} \;\; \forall j \in \{1,2\}
\end{aligned}
\tag{14}
$$

The optimal value of t: -5.306
Verification result: Inconclusive

9.2 MILP formulation with RaVeN layerwise constraints on the illustrative example

We show the layerwise formulation of RaVeN with the concrete bounds from the DeepZ analysis. We use the optimal neuron-level convex relaxation (triangle relaxation) for the ReLU activation. For example, the linear constraints for ReLU assignment $x_5^1 \leftarrow ReLU(x_3^1)$ are shown below.

$$0 \le x_5^1,\ x_3^1 \le x_5^1,\ x_5^1 \le \frac{5}{8} \cdot x_3^1 + \frac{45}{8}, x_5^1 \le 15$$

Similar to the approach in [88], the cross-execution constraints (highlighted in blue) are only applied at the input layer. However, the RaVeN layerwise approach more effectively preserves linear relationships across multiple executions. For instance, using constraints like $(x_1^1 - x_1^2) = 3$, $(x_2^1 - x_2^2) = -3$, and $x_3^1 = x_1^1 - x_2^1, x_3^2 = x_1^2 - x_2^2$, the layerwise formulation can deduce that $(x_3^1 - x_3^2) = 6$. Nevertheless, the layerwise approach loses precision in tracking dependencies beyond activation layers (e.g., ReLU, Sigmoid) due to convex overapproximation. This is why we require a DiffPoly analysis with custom abstract transformers explicitly designed for difference tracking. The optimal value of t and the verification result for this formulation is shown below.

$$\begin{aligned}
\min\ \ & t \\
& \text{subject to} \\
& min(F_1) = z_1, z_1 \le t, min(F_2) = z_2, z_2 \le t \quad \text{[MILP encoding of } \Psi] \\
& F_1 = x_7^1 - x_8^1, F_2 = -x_7^2 + x_8^2 \\
& x_8^1 = -x_5^1 + x_6^1, -15\frac{35}{36} \le x_8^2 \le 6\frac{5}{8}, x_8^2 = -x_5^2 + x_6^2, -16\frac{2}{9} \le x_8^2 \le 15\frac{5}{8} \\
& x_7^1 = x_5^1 - x_6^1, -6\frac{5}{8} \le x_7^1 \le 15\frac{35}{36}, x_7^2 = x_5^2 - x_6^2, -15\frac{5}{8} \le x_7^2 \le 16\frac{2}{9} \\
& x_4^1 \le x_6^1 \le \frac{1}{36} \cdot x_4^1 + \frac{35}{36}, 0 \le x_6^1 \le 1, x_4^2 \le x_6^2 \le \frac{5}{18} \cdot x_4^2 + \frac{65}{9}, 0 \le x_6^2 \le 10 \\
& x_3^1 \le x_5^1 \le \frac{5}{8} \cdot x_3^1 + \frac{45}{8}, 0 \le x_5^1 \le 15, x_3^2 \le x_5^2 \le \frac{3}{8} \cdot x_3^2 + \frac{45}{8}, 0 \le x_5^2 \le 9 \\
& x_4^1 = -2 \cdot x_1^1 + x_2^1, -35 \le x_4^1 \le 1, x_4^2 = -2 \cdot x_1^2 + x_2^2, -26 \le x_4^2 \le 10 \\
& x_3^1 = x_1^1 - x_2^1, -9 \le x_3^1 \le 15, x_3^2 = x_1^2 - x_2^2, -15 \le x_3^2 \le 9 \\
& (x_1^1 - x_1^2) = 3, (x_2^1 - x_2^2) = -3 \quad \text{[cross-execution constraints at input layer]} \\
& 8 \le x_1^1 \le 20, 5 \le x_2^1 \le 17, 5 \le x_1^2 \le 17, 8 \le x_2^2 \le 20
\end{aligned} \tag{15}$$

The optimal value of t: −1.564
Verification result: Inconclusive

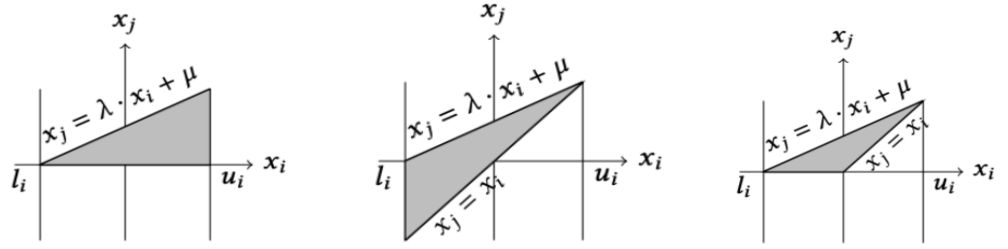
9.3 MILP Formulation of RaVeN with difference tracking for Illustrative Example

We show the MILP formulation obtained by adding the difference constraints (shown in blue) obtained from DiffPoly analysis to the layerwise formulation (Eq. 15). The optimal value of t and the verification result for this formulation is shown below.

$$
\begin{aligned}
\min\ \ & t \\
& \text{subject to} \\
& min(F_1) = z_1, z_1 \leq t, min(F_2) = z_2, z_2 \leq t \quad \text{[MILP encoding of } \Psi\text{]} \\
& F_1 = x_7^1 - x_8^1, F_2 = -x_7^2 + x_8^2 \\
& x_8^1 = -x_5^1 + x_6^1, -15\frac{35}{36} \leq x_8^2 \leq 6\frac{5}{8}, x_8^2 = -x_5^2 + x_6^2, -16\frac{2}{9} \leq x_8^2 \leq 15\frac{5}{8} \\
& x_7^1 = x_5^1 - x_6^1, -6\frac{5}{8} \leq x_7^1 \leq 15\frac{35}{36}, x_7^2 = x_5^2 - x_6^2, -15\frac{5}{8} \leq x_7^2 \leq 16\frac{2}{9} \\
& x_4^1 \leq x_6^1 \leq \frac{1}{36} \cdot x_4^1 + \frac{35}{36}, 0 \leq x_6^1 \leq 1, x_4^2 \leq x_6^2 \leq \frac{5}{18} \cdot x_4^2 + \frac{65}{9}, 0 \leq x_6^2 \leq 10 \\
& x_3^1 \leq x_5^1 \leq \frac{5}{8} \cdot x_3^1 + \frac{45}{8}, 0 \leq x_5^1 \leq 15, x_3^2 \leq x_5^2 \leq \frac{3}{8} \cdot x_3^2 + \frac{45}{8}, 0 \leq x_5^2 \leq 9 \\
& x_4^1 = -2 \cdot x_1^1 + x_2^1, -35 \leq x_4^1 \leq 1, x_4^2 = -2 \cdot x_1^2 + x_2^2, -26 \leq x_4^2 \leq 10 \\
& x_3^1 = x_1^1 - x_2^1, -9 \leq x_3^1 \leq 15, x_3^2 = x_1^2 - x_2^2, -15 \leq x_3^2 \leq 9 \\
& 8 \leq x_1^1 \leq 20, 5 \leq x_2^1 \leq 17, 5 \leq x_1^2 \leq 17, 8 \leq x_2^2 \leq 20 \\
& \delta_1^{1,2} = x_1^1 - x_1^2, 3 \leq \delta_1^{1,2} \leq 3 \\
& \delta_2^{1,2} = x_2^1 - x_2^2, -3 \leq \delta_2^{1,2} \leq -3 \\
& \delta_1^{1,2} - \delta_2^{1,2} \leq \delta_3^{1,2} \leq \delta_1^{1,2} - \delta_2^{1,2} \\
& \delta_3^{1,2} = x_3^1 - x_3^2, 6 \leq \delta_3^{1,2} \leq 6 \\
& -2 \cdot \delta_1^{1,2} + \delta_2^{1,2} \leq \delta_4^{1,2} \leq -2 \cdot \delta_1^{1,2} + \delta_2^{1,2} \\
& \delta_4^{1,2} = x_4^1 - x_4^2, -9 \leq \delta_4^{1,2} \leq -9 \\
& \delta_5^{1,2} = x_5^1 - x_5^2, 0 \leq \delta_5^{1,2} \leq \delta_3^{1,2}, 0 \leq \delta_5^{1,2} \leq 6 \\
& \delta_6^{1,2} = x_6^1 - x_6^2, \delta_4^{1,2} \leq \delta_6^{1,2} \leq 0, -9 \leq \delta_6^{1,2} \leq 0 \\
& \delta_7^{1,2} = x_7^1 - x_7^2, 0 \leq \delta_7^{1,2} \leq 15 \\
& \delta_8^{1,2} = x_8^1 - x_8^2, -15 \leq \delta_8^{1,2} \leq 0
\end{aligned}
\tag{16}
$$

The optimal value of t: 0.0
Verification result: UAP does not exist

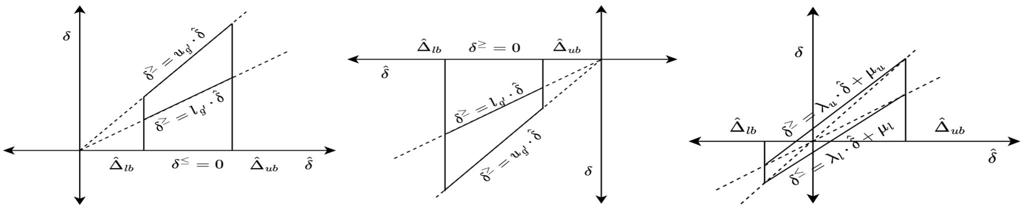
10 CONVEX RELAXATION OF RELU


x_j
$x_j = \lambda \cdot x_i + \mu$
l_i
u_i
x_i
x_j
$x_j = \lambda \cdot x_i + \mu$
$x_j = x_i$
l_i
u_i
x_i
x_j
$x_j = \lambda \cdot x_i + \mu$
$x_j = x_i$
l_i
u_i
x_i

(a) $u_{x_i} < -l_{x_i}$ (b) $u_{x_i} \geq -l_{x_i}$ (c) optimal

Fig. 13. The convex approximations for $x_j = ReLU(x_i)$ where $x_i \in [l_{x_i}, u_{x_i}]$ and $(l_{x_i} < 0) \wedge (u_{x_i} > 0)$. The

11 DIFFPOLY TRANSFORMER FOR DIFFERENTIABLE ACTIVATIONS


δ
$\delta^{\geq} = u_{g'} \cdot \hat{\delta}$
$\delta^{\geq} = l_{g'} \cdot \hat{\delta}$
$\hat{\Delta}_{lb}$
$\delta^{\leq} = 0$
$\hat{\Delta}_{ub}$
$\hat{\delta}$
$\hat{\Delta}_{lb}$
$\delta^{\geq} = 0$
$\hat{\Delta}_{ub}$
$\hat{\delta}$
$\delta^{\geq} = l_{g'} \cdot \hat{\delta}$
$\delta^{\geq} = u_{g'} \cdot \hat{\delta}$
δ
δ
$\delta^{\geq} = \lambda_u \cdot \hat{\delta} + \mu_u$
$\hat{\Delta}_{lb}$
$\hat{\Delta}_{ub}$
$\delta^{\leq} = \lambda_l \cdot \hat{\delta} + \mu_l$
$\hat{\delta}$

(a) $\hat{\Delta}_{lb} \geq 0$ (b) $\hat{\Delta}_{ub} \leq 0$ (c) $\hat{\Delta}_{lb} < 0 \wedge \hat{\Delta}_{ub} > 0$

Fig. 14. The optimal (in terms of area) convex approximations for $\delta = g(x) - g(y)$ where $\hat{\delta} = (x - y)$, $\delta^{\geq}$, $\delta^{\leq}$ are symbolic upper bound and lower bound of δ respectively and where g is a differentiable activation function.

12 PSEUDOCODE FOR BACK-SUBSTITUTION ALGORITHM

Algorithm 2 Back-substitution Algorithm

```
procedure BACK-SUBSTITUTION($\delta_{x_i}^{a,b,\leq}, \delta_{x_i}^{a,b,\geq}$, a ∈ $\mathcal{A}_{2i}$)
    Input: $\delta_{x_i}^{a,b,\leq}, \delta_{x_i}^{a,b,\geq}$, a ∈ $\mathcal{A}_{2i}$
    Output: $\Delta_{lb}^{a,b,x_i}, \Delta_{ub}^{a,b,x_i}$
    $\Delta_{lb}^{a,b,x_i} \leftarrow -\infty$;   $\Delta_{ub}^{a,b,x_i} \leftarrow \infty$
    while $True$ do
        $t_{\Delta_{lb}} \leftarrow S_c(\delta_{x_i}^{a,b,\leq}$, a)      ▷ the concrete bounds required for concrete substitution are in a
        $t_{\Delta_{ub}} \leftarrow S_c(\delta_{x_i}^{a,b,\geq}$, a)      ▷ the concrete bounds required for concrete substitution are in a
        $\Delta_{lb}^{a,b,x_i} \leftarrow \max(\Delta_{lb}^{a,b,x_i}, t_{\Delta_{lb}})$; $\Delta_{ub}^{a,b,x_i} \leftarrow \min(\Delta_{ub}^{a,b,x_i}, t_{\Delta_{ub}})$
        if $\delta_{x_i}^{a,b,\leq}$ and $\delta_{x_i}^{a,b,\geq}$ have only input variables then
            break;
        end if
        $\delta_{x_i}^{a,b,\leq} \leftarrow S_s(\delta_{x_i}^{a,b,\leq}$, a)      ▷ the symbolic bounds required for symbolic substitution are in a
        $\delta_{x_i}^{a,b,\geq} \leftarrow S_c(\delta_{x_i}^{a,b,\geq}$, a)      ▷ the symbolic bounds required for symbolic substitution are in a
    end while
end procedure
return $\Delta_{lb}^{a,b,x_i}, \Delta_{ub}^{a,b,x_i}$;
```

LEMMA 12.1. *If* $(\delta_{x_i}^{a,b,\leq} \leq \delta_{x_i}^{a,b}) \wedge (\delta_{x_i}^{a,b} \leq \delta_{x_i}^{a,b,\geq})$ *then the concrete lower* Δ_{lb}^{a,b,x_i} *and concrete upper bound* Δ_{ub}^{a,b,x_i} *obtained with Back-Substitution on symbolic bounds* $\delta_{x_i}^{a,b,\leq}$ *and* $\delta_{x_i}^{a,b,\geq}$ *then* $\Delta_{lb}^{a,b,x_i} \leq \delta_{x_i}^{a,b}$ *and* $\delta_{x_i}^{a,b} \leq \Delta_{ub}^{a,b,x_i}$ *holds.*

PROOF. For the proof refer to Theorem 4.9 of [69]. □

13 SOUNDNESS OF RAVEN

In this section, we formally prove the soundness of RaVeN. We first show the soundness of the abstract transformers of DiffPoly.

13.1 Soundness Proof of the DiffPoly ReLU transformer

Theorem 4.4. *(Soundness of DiffPoly Relu Transformer) For any abstract element* $\overline{a} \in \mathcal{A}_{2i}$ $T_R(\gamma_{2i}(\overline{a})) \subseteq \gamma_{2i+2}(T_R^{\sharp}(\overline{a}))$.

Proof. For any $(X^a, X^b) \in \gamma_{2i}(\overline{a})$ we denote $ReLU(x_i^a) = y_i^a$ and $ReLU(x_i^b) = y_i^b$ where $X^a = [x_1^a, \dots, x_i^a]^T \in \mathbb{R}^i$, $X^b = [x_1^b, \dots, x_i^b]^T \in \mathbb{R}^i$. We use $\delta_{y_i}^{a,b}$ to denote the difference $\delta_{y_i}^{a,b} = y_i^a - y_i^b$. For any element $\overline{a} \in \mathcal{A}_{2i}$, $\overline{a'} = T_R^{\sharp}(\overline{a})$ where $\overline{a'} = [a_1, \dots, a_i, a'_{i+1}]$ and $a'_i =< C'^{i+1}_{sym}, C'^{i+1}_{con} >$ constructed as described in Section 4.2. C'^{i+1}_{sym} and C'^{i+1}_{con} given by

$$C'^{i+1}_{sym} =< y_i^{a,\leq}, y_i^{b,\leq}, \delta_{y_i}^{a,b,\leq}, y_i^{a,\geq}, y_i^{b,\geq}, \delta_{y_i}^{a,b,\geq} > \qquad C'^{i+1}_{con} =< l_{a,y_i}, l_{b,y_i}, \Delta_{lb}^{a,b,y_i}, u_{a,y_i}, u_{b,y_i}, \Delta_{ub}^{a,b,y_i} >$$

We use symbolic bounds of $y_i^{a,\leq}, y_i^{a,\geq}$ and $y_i^{b,\leq}, y_i^{b,\geq}$ of y_i^a, y_i^b described in existing work [69, 92]. For the correctness of symbolic bounds, $y_i^{a,\leq}, y_i^{a,\geq}$ and $y_i^{b,\leq}, y_i^{b,\geq}$ we only the state the results and refer the readers to [69, 92] for details.

$$\begin{aligned}
&\forall (X^a, X^b) \in \gamma_{2i}(\overline{a}).\ (y_i^a = ReLU(x_i^a)) \wedge (y_i^b = ReLU(x_i^b)) \\
&\implies \forall j \in [i].\ (x_j^a \in [l_{a,x_j}, u_{a,x_j}]) \wedge (x_j^b \in [l_{b,x_j}, u_{b,x_j}]) \\
&\implies \forall j \in [i].\ (x_j^{a,\leq} \leq x_j^a) \wedge (x_j^a \leq x_j^{a,\geq}) \wedge (x_j^{b,\leq} \leq x_j^b) \wedge (x_j^b \leq x_j^{b,\geq}) \\
&\implies (y_i^{a,\leq} \leq y_i^a) \wedge (y_i^a \leq y_i^{a,\geq}) \wedge (y_i^{b,\leq} \leq y_i^b) \wedge (y_i^b \leq y_i^{b,\geq})
\end{aligned} \tag{17}$$

From Theorem 3.2 in [92] and Theorem 4.2 in [69]

$$\implies \quad (y_i^a \in [l_{a,y_i}, u_{a,y_i}]) \wedge (y_i^b \in [l_{b,y_i}, u_{b,y_i}]) \ \text{ From Lemma 4.3} \tag{18}$$

$$\begin{aligned}
&\forall (X^a, X^b) \in \gamma_{2i}(\overline{a}).\ (y_i^a = ReLU(x_i^a)) \wedge (y_i^b = ReLU(x_i^b)) \\
&\implies \forall j \in [i].\ (x_j^a \in [l_{a,x_j}, u_{a,x_j}]) \wedge (x_j^b \in [l_{b,x_j}, u_{b,x_j}]) \wedge (\delta_{x_j}^{a,b} \in [\Delta_{lb}^{a,b,x_j}, \Delta_{ub}^{a,b,x_j}]) \\
&\implies (\delta_{y_i}^{a,b,\leq} \leq \delta_{y_i}^{a,b}) \wedge (\delta_{y_i}^{a,b} \leq \delta_{y_i}^{a,b,\geq}) \wedge (\delta_{x_i}^{a,b} \in [\Delta_{lb}^{a,b,y_i}, \Delta_{ub}^{a,b,y_i}]) \ \text{ From Lemma 4.2 and 4.3}
\end{aligned} \tag{19}$$

From 17, 18 and 19 we show that

$$\begin{aligned}
&\forall (X^a, X^b) \in \gamma_{2i}(\overline{a}).\ (y_i^a = ReLU(x_i^a)) \wedge (y_i^b = ReLU(x_i^b)) \\
&\qquad \implies ([x_1^a, \dots, x_i^a, y_i^a]^T, [x_1^b, \dots, x_i^b, y_i^b)]^T) \in \gamma_{2i+2}(\overline{a'})
\end{aligned} \tag{20}$$

□

Eq. 20 proves that $T_R(\gamma_{2i}(\overline{a})) \subseteq \gamma_{2i+2}(T_R^{\sharp}(\overline{a}))$

13.2 Soundness Proof of the DiffPoly transformer for differentiable activations

We first state the lemmas required to prove the soundness of $T_g^{\sharp}$ where g represents differentiable activation functions such as Sigmoid and Tanh. Proofs of the lemmas 13.1, 13.2 are in Appendix 14.2.

Lemma 13.1. *(Correctness of symbolic bounds in Table 4) If* $x_i^a \in [l_{a,x_i}, u_{a,x_i}]$, $x_i^b \in [l_{b,x_i}, u_{b,x_i}]$ *and* $\delta_{x_i}^{a,b} = (x_i^a - x_i^b) \in [\Delta_{lb}^{a,b,x_i}, \Delta_{ub}^{a,b,x_i}]$ *and* $\delta_{y_i}^{a,b} = g(x_i^a) - g(x_i^b)$ *then* $\delta_{y_i}^{a,b,\leq} \leq \delta_{y_i}^{a,b} \leq \delta_{y_i}^{a,b,\geq}$ *where* $\delta_{y_i}^{a,b,\leq}$ *and* $\delta_{y_i}^{a,b,\geq}$ *defined in Table 4.*

Lemma 13.2. *(Correctness of concrete bounds computed by* $T_g^{\sharp}$*) If* $x_i^a \in [l_{a,x_i}, u_{a,x_i}]$, $x_i^b \in [l_{b,x_i}, u_{b,x_i}]$ *and* $\delta_{x_i}^{a,b} = (x_i^a - x_i^b) \in [\Delta_{lb}^{a,b,x_i}, \Delta_{ub}^{a,b,x_i}]$, $y_i^a = g(x_i^a)$, $y_i^b = g(x_i^b)$, $\delta_{y_i}^{a,b} = y_i^a - y_i^b$ *then* $l_{a,y_i} \leq y_i^a \leq u_{a,y_i}$,

$l_{b,y_i} \leq y_i^b \leq u_{b,y_i}$, and $\Delta_{lb}^{a,b,y_i} \leq \delta_{y_i}^{a,b} \leq \Delta_{ub}^{a,b,y_i}$ where Δ_{lb}^{a,b,y_i} and Δ_{ub}^{a,b,y_i} computed by applying back-substitution on $\delta_{y_i}^{a,b,\leq}$ and $\delta_{y_i}^{a,b,\geq}$ respectively.

The concrete transformer $T_g : \wp(\mathbb{R}^{2i}) \to \wp(\mathbb{R}^{2i+2})$ for the assignments $y_i^a \leftarrow g(x_i^a), y_i^b \leftarrow g(x_i^b)$ is defined as $T_g(\mathcal{X}) = \{([x_1^a,\dots,x_i^a,y_i^a]^T,[x_1^b,\dots,x_i^b,y_i^b]^T) \mid (X^a,X^b) \in \mathcal{X}\}$ where $y_i^a = g(x_i^a)$, $y_i^b = g(x_i^b)$, $\mathcal{X} \subseteq \mathbb{R}^{2i}$ and $X^a = [x_1^a,\dots,x_i^a]^T \in \mathbb{R}^i$, $X^b = [x_1^b,\dots,x_i^b]^T \in \mathbb{R}^i$.

Theorem 13.3 (Soundness of DiffPoly Sigmoid and Tanh Transformer). *For any abstract element $\overline{a} \in \mathcal{A}_{2i}$ $T_g(\gamma_{2i}(\overline{a})) \subseteq \gamma_{2i+2}(T_g^\sharp(\overline{a}))$.*

Proof. For any $(X^a,X^b) \in \gamma_{2i}(\overline{a})$ we denote $g(x_i^a) = y_i^a$ and $g(x_i^b) = y_i^b$ where $X^a = [x_1^a,\dots,x_i^a]^T \in \mathbb{R}^i$, $X^b = [x_1^b,\dots,x_i^b]^T \in \mathbb{R}^i$. We use $\delta_{y_i}^{a,b}$ to denote the difference $\delta_{y_i}^{a,b} = y_i^a - y_i^b$. For any element $\overline{a} = [a_1,\dots,a_i] \in \mathcal{A}_{2i}, \overline{a'} = T_g^\sharp(\overline{a})$ where $\overline{a'} = [a_1,\dots,a_i,a'_{i+1}]$ and $a'_{i+1} =< C'^{i+1}_{sym}, C'^{i+1}_{con} >$ constructed as described in Section 4.3. C'^{i+1}_{sym} and C'^{i+1}_{con} given by

$$C'^{i+1}_{sym} =< y_i^{a,\leq}, y_i^{b,\leq}, \delta_{y_i}^{a,b,\leq}, y_i^{a,\geq}, y_i^{b,\geq}, \delta_{y_i}^{a,b,\geq} > \qquad C'^{i+1}_{con} =< l_{a,y_i}, l_{b,y_i}, \Delta_{lb}^{a,b,y_i}, u_{a,y_i}, u_{b,y_i}, \Delta_{ub}^{a,b,y_i} >$$

We use symbolic bounds of $y_i^{a,\leq}, y_i^{a,\geq}$ and $y_i^{b,\leq}, y_i^{b,\geq}$ of y_i^a, y_i^b described in existing work [69]. For the correctness of symbolic bounds, $y_i^{a,\leq}, y_i^{a,\geq}$ and $y_i^{b,\leq}, y_i^{b,\geq}$ we only the state the results and refer the readers to [69] for details.

$$\begin{aligned}
&\forall (X^a,X^b) \in \gamma_{2i}(\overline{a}).\ \ (y_i^a = g(x_i^a)) \wedge (y_i^b = g(x_i^b)) \\
&\implies \forall j \in [i].\ \ (x_j^a \in [l_{a,x_j}, u_{a,x_j}]) \wedge (x_j^b \in [l_{b,x_j}, u_{b,x_j}]) \\
&\implies \forall j \in [i].\ \ (x_j^{a,\leq} \leq x_j^a) \wedge (x_j^a \leq x_j^{a,\geq}) \wedge (x_j^{b,\leq} \leq x_j^b) \wedge (x_j^b \leq x_j^{b,\geq}) \\
&\implies (y_i^{a,\leq} \leq y_i^a) \wedge (y_i^a \leq y_i^{a,\geq}) \wedge (y_i^{b,\leq} \leq y_i^b) \wedge (y_i^b \leq y_i^{b,\geq}) && (21)\\
&\text{From Theorem 4.3 [69]} && (22)\\
&\implies \quad (y_i^a \in [l_{a,y_i}, u_{a,y_i}]) \wedge (y_i^b \in [l_{b,y_i}, u_{b,y_i}])\ \ \text{From Lemma 13.2} && (23)
\end{aligned}$$

$$\begin{aligned}
&\forall (X^a,X^b) \in \gamma_{2i}(\overline{a}).\ \ (y_i^a = g(x_i^a)) \wedge (y_i^b = g(x_i^b)) \\
&\implies \forall j \in [i].\ \ (x_j^a \in [l_{a,x_j}, u_{a,x_j}]) \wedge (x_j^b \in [l_{b,x_j}, u_{b,x_j}]) \wedge (\delta_{x_j}^{a,b} \in [\Delta_{lb}^{a,b,x_j}, \Delta_{ub}^{a,b,x_j}]) \\
&\implies (\delta_{y_i}^{a,b,\leq} \leq \delta_{y_i}^{a,b}) \wedge (\delta_{y_i}^{a,b} \leq \delta_{y_i}^{a,b,\geq}) \wedge (\delta_{x_i}^{a,b} \in [\Delta_{lb}^{a,b,y_i}, \Delta_{ub}^{a,b,y_i}])\ \ \text{From Lemma 13.1 and 13.2} && (24)
\end{aligned}$$

From 21, 23 and 24 we show that

$$\begin{aligned}
&\forall (X^a,X^b) \in \gamma_{2i}(\overline{a}).\ \ (y_i^a = g(x_i^a)) \wedge (y_i^b = g(x_i^b)) \\
&\implies ([x_1^a,\dots,x_i^a,y_i^a]^T,[x_1^b,\dots,x_i^b,y_i^b]^T) \in \gamma_{2i+2}(\overline{a'}) && (25)
\end{aligned}$$

Eq. 25 proves that $(\gamma_{2i}(\overline{a})) \subseteq \gamma_{2i+2}(T_g^\sharp(\overline{a}))$. □

13.3 Soundness Proof of the DiffPoly Affine Transformer

First, we describe the concrete affine transformer $T_A : \wp(\mathbb{R}^{2i}) \to \wp(\mathbb{R}^{2i+2})$. Let, $W \in \mathbb{R}^i$ and $v_{i+1} \in \mathbb{R}$ denote the weight vector and bias respectively then the concrete transformer is given below where $x_{i+1}^a = v + \sum_{j=1}^i w_j \cdot x_j^a$ and $x_{i+1}^b = v + \sum_{j=1}^i w_j \cdot x_j^b$

$$T_A(\mathcal{X}) = \{([x_1^a,\dots,x_i^a,x_{i+1}^a]^T,[x_1^b,\dots,x_i^b,x_{i+1}^b)]^T) \mid (X^a,X^b) \in \mathcal{X}\}$$

We first state a couple of lemmas needed to prove the soundness of $T_A^\sharp$. The proof of the lemmas 13.4 and 13.5 is in Appendix 14.3.

Lemma 13.4. *(Correctness of symbolic bounds computed by the affine transformer) If* $\forall j \in [i].\ x^a_j \in [l_{a,x_j}, u_{a,x_j}]$, $\forall j \in [i].\ x^b_j \in [l_{b,x_j}, u_{b,x_j}]$ *and* $\forall j \in [i].\ \delta^{a,b}_{x_j} \in [\Delta^{a,b,x_j}_{lb}, \Delta^{a,b,x_j}_{ub}]$ *and* $x^a_{i+1} = v + \sum_{j=1}^{i} w_j \cdot x^a_j$, $x^b_{i+1} = v + \sum_{j=1}^{i} w_j \cdot x^b_j$, *and* $\delta^{a,b}_{x_{i+1}} = (x^a_{i+1} - x^b_{i+1})$ *then* $x^{a,\leq}_{i+1} \leq x^a_{i+1} \leq x^{a,\geq}_{i+1}$, $x^{b,\leq}_{i+1} \leq x^b_{i+1} \leq x^{b,\geq}_{i+1}$ *and* $\delta^{a,b,\leq}_{x_{i+1}} \leq \delta^{a,b}_{x_{i+1}} \leq \delta^{a,b,\geq}_{x_{i+1}}$ *where* $x^{a,\leq}_{i+1}, x^{a,\geq}_{i+1}, x^{b,\leq}_{i+1}, x^{b,\geq}_{i+1}, \delta^{a,b,\leq}_{x_{i+1}}$ *and* $\delta^{a,b,\geq}_{x_{i+1}}$ *defined in Eq. 8.*

Lemma 13.5. *(Correctness of concrete bounds computed by the affine transformer) If* $\forall j \in [i].\ x^a_j \in [l_{a,x_j}, u_{a,x_j}]$, $\forall j \in [i].\ x^b_j \in [l_{b,x_j}, u_{b,x_j}]$ *and* $\forall j \in [i].\ \delta^{a,b}_{x_j} \in [\Delta^{a,b,x_j}_{lb}, \Delta^{a,b,x_j}_{ub}]$ *and* $x^a_{i+1} = v + \sum_{j=1}^{i} w_j \cdot x^a_j$, $x^b_{i+1} = v + \sum_{j=1}^{i} w_j \cdot x^b_j$, *and* $\delta^{a,b}_{x_{i+1}} = (x^a_{i+1} - x^b_{i+1})$ *then* $l_{a,x_{i+1}} \leq x^a_{i+1} \leq u_{a,x_{i+1}}$, $l_{b,x_{i+1}} \leq x^b_{i+1} \leq u_{b,x_{i+1}}$ *and* $\Delta^{a,b,x_{i+1}}_{lb} \leq \delta^{a,b}_{x_{i+1}} \leq \Delta^{a,b,x_{i+1}}_{ub}$.

Theorem 13.6. *(Soundness of DiffPoly Affine Transformer) For all abstract element* $\overline{a} \in \mathcal{A}_{2i}$ $T_A(\gamma_{2i}(\overline{a})) \subseteq \gamma_{2i+2}(T^\sharp_A(\overline{a}))$.

Proof. For any $(X^a, X^b) \in \gamma_{2i}(\overline{a})$ we denote $x^a_{i+1} = v + \sum_{j=1}^{i} w_j \cdot x^a_j$, $x^b_{i+1} = v + \sum_{j=1}^{i} w_j \cdot x^b_j$ where $W \in \mathbb{R}^i$ is the weight vector, $v \in \mathbb{R}$ is the bias vector and $\delta^{a,b}_{x_{i+1}} = (x^a_{i+1} - y^b_{i+1})$. For any element $\overline{a} \in \mathcal{A}_{2i}$, $\overline{a'} = T^\sharp_A(\overline{a})$ where $\overline{a'} = [a_1, \dots, a_i, a'_{i+1}]$ and $a'_{i+1} =< C'^{i+1}_{sym}, C'^{i+1}_{con} >$ constructed as described in Section 4.4. C'^{i+1}_{sym} and C'^{i+1}_{con} given by

$$C'^{i+1}_{sym} =< x^{a,\leq}_{i+1}, x^{b,\leq}_{i+1}, \delta^{a,b,\leq}_{x_{i+1}}, x^{a,\geq}_{i+1}, x^{b,\geq}_{i+1}, \delta^{a,b,\geq}_{x_{i+1}} > \qquad C'^{i}_{con} =< l_{a,x_{i+1}}, l_{b,x_{i+1}}, \Delta^{a,b,x_{i+1}}_{lb}, u_{a,x_{i+1}}, u_{b,x_{i+1}}, \Delta^{a,b,x_{i+1}}_{ub} >$$

$$\begin{aligned}
&\forall (X^a, X^b) \in \gamma_{2i}(\overline{a}). \quad (x^a_{i+1} = v + \sum_{j=1}^{i} w_j \cdot x^a_j) \land (x^b_{i+1} = v + \sum_{j=1}^{i} w_j \cdot x^b_j) \\
&\implies \forall i \in [i].\ (x^a_j \in [l_{a,x_j}, u_{a,x_j}]) \land (x^b_j \in [l_{b,x_i}, u_{b,x_j}]) \\
&\implies (x^{a,\leq}_{i+1} \leq x^a_{i+1}) \land (x^a_{i+1} \leq x^{a,\geq}_{i+1}) \land (x^{b,\leq}_{i+1} \leq x^b_{i+1}) \land (x^b_{i+1} \leq x^{b,\geq}_{i+1}) \ \text{From Lemma 13.4} \qquad (26) \\
&\implies (x^a_{i+1} \in [l_{a,x_{i+1}}, u_{a,x_{i+1}}]) \land (x^b_{i+1} \in [l_{b,x_{i+1}}, u_{b,x_{i+1}}]) \ \text{From Lemma 13.5} \qquad (27)
\end{aligned}$$

$$\begin{aligned}
&\forall (X^a, X^b) \in \gamma_{2i}(\overline{a}).\ (x^a_{i+1} = v + \sum_{j=1}^{i} w_j \cdot x^a_j) \land (x^b_{i+1} = v + \sum_{j=1}^{i} w_j \cdot x^b_j) \\
&\implies \forall j \in [i].\ (x^a_j \in [l_{a,x_j}, u_{a,x_j}]) \land (x^b_j \in [l_{b,x_j}, u_{b,x_j}]) \land (\delta^{a,b}_{x_j} \in [\Delta^{a,b,x_j}_{lb}, \Delta^{a,b,x_j}_{ub}]) \\
&\implies (\delta^{a,b,\leq}_{x_{i+1}} \leq \delta^{a,b}_{x_{i+1}}) \land (\delta^{a,b}_{x_{i+1}} \leq \delta^{a,b,\geq}_{x_{i+1}}) \land (\delta^{a,b}_{x_{i+1}} \in [\Delta^{a,b,x_{i+1}}_{lb}, \Delta^{a,b,x_{i+1}}_{ub}]) \ \text{From Lemma 13.4 and 13.5} \qquad (28)
\end{aligned}$$

From 26, 27 and 28 we show that

$$\begin{aligned}
&\forall (X^a, X^b) \in \gamma_{2i}(\overline{a}).\ (x^a_{i+1} = v + \sum_{j=1}^{i} w_j \cdot x^a_j) \land (x^b_{i+1} = v + \sum_{j=1}^{i} w_j \cdot x^b_j) \qquad (29) \\
&\implies ([x^a_1, \dots, x^a_i, x^a_{i+1}]^T, [x^b_1, \dots, x^b_i, x^b_{i+1})]^T) \in \gamma_{2n}(\overline{a'}) \qquad (30)
\end{aligned}$$

Eq. 30 shows that $T_A(\gamma_{2i}(\overline{a})) \subseteq \gamma_{2i+2}(T^\sharp_A(\overline{a'}))$ □

13.4 Soundness Proof of Product DNN analysis

Theorem 4.5. *(Soundness of Product DNN analysis)* $\forall (X_1, \dots, X_k) \in \mathbb{R}^{n_0 \times k}.\Phi((X_1, \dots, X_k)) \implies (\mathcal{N}^k((X_1, \dots, X_k)) \in \mathbb{P})$.

Proof. $\mathbb{P} = \bigtimes_{i=1}^{k} \mathcal{P}_i$ implies $(Y_1, \dots, Y_k) \in \mathbb{P} \iff \wedge_{i=1}^{k} (Y_i \in \mathcal{P}_i)$ where $\forall i \in [k].(Y_i \in \mathbb{R}^{n_l})$.

$$\begin{aligned} \forall X_1, \dots, X_k \in \mathbb{R}^{n_0}.\Phi((X_1, \dots, X_k)) &\implies \wedge_{i=1}^{k} \phi_{in}^{i}(X_i) \implies \wedge_{i=1}^{k} (N(X_i) \in \mathcal{P}_i) \\ &\implies [N(X_1) \dots, N(X_k)]^T \in \mathbb{P} \implies \mathcal{N}^k((X_1, \dots, X_k)) \in \mathbb{P} \end{aligned}$$

□

13.5 Soundness of RaVeN LP Formulation

Theorem 4.6. *(Soundness of Linear constraints)* $\Phi_t \subseteq \mathcal{L}_t^0$ *and* $\forall i \in [l].\forall X_1, \dots X_k \in \mathbb{R}^{n_0}$. $\Phi(X_1, \dots, X_k) \implies (N^i(X_1), \dots, N^i(X_k)) \in \mathcal{L}_t^i$ *where* $N^i : \mathbb{R}^{n_0} \to \mathbb{R}^{n_i}$ *is the composition of first i layers of the network N,* $N^i = N_1 \circ \dots \circ N_i$.

Proof. The input specification Φ is defined as a set of linear constraints over the input variables and exactly encoded as a set of linear constraints. Hence, $\mathcal{L}_t^0$ is same as Φ_t, $\mathcal{L}_t^0 = \Phi_t$. $\forall i \in [l]$ $\mathcal{L}^i$ is defined from the constraints in Eq 9. We show that all concrete bounds $l_j^{a,l}, u_j^{a,l}, \Delta_{lb}^{a,b,l,x_j}, \Delta_{ub}^{a,b,l,x_j}$ and all symbolic bounds $x_j^{a,l,\leq}, x_j^{a,l,\geq}, \delta_j^{a,b,l,\leq}, \delta_j^{a,b,\geq}$ shown in 9. From Lemma 4.3, 13.2 and,13.5 all concrete bounds satisfy Eq 9. From Lemma , 13.1, 4.2, and, 13.4 all symbolic bounds satisfy Eq 9. □

13.6 Correctness of encoding of Ψ

The output specification $\Psi : \mathbb{R}^{n_l \times k} \to \{True, False\}$ is defined as $\Psi(Y_1, \dots, Y_k) = \bigwedge_{i=1}^{m} \left(\bigvee_{j=1}^{n} \psi_{i,j}(Y_1, \dots, Y_k) \right)$, $\psi_{i,j}(Y_1, \dots, Y_k) = \left(\sum_{i'=1}^{k} C_{i,j,i'}^T Y_{i'} \geq 0 \right)$ and $C_{i,j,i'} \in \mathbb{R}^{n_l}$. We show that the following objective computes the minimum number of clauses that remain satisfied for all $(Y_1, \dots, Y_k)$.

$$\min_{(Y_1,\dots,Y_k)} \sum_{i=1}^{m} z_i \quad \text{s.t.} \quad x_{i,j} = \psi_{i,j}(Y_1, \dots, Y_k) = \left(\sum_{i'=1}^{k} C_{i,j,i'}^T Y_{i'} \geq 0 \right); z_i = \left(\sum_{j=1}^{n} x_{i,j} \geq 0 \right) \tag{31}$$

For any $(Y_1, \dots, Y_k)$ for all $i \in [m]$ and $j \in [n]$ $(x_{i,j} = 1) \iff \left(\sum_{i'=1}^{k} C_{i,j,i'}^T Y_{i'} \geq 0 \right)$. Then $\left(\sum_{j=1}^{n} x_{i,j} \geq 0 \right) \iff \bigvee_{j=1}^{n} \psi_{i,j}(Y_1, \dots, Y_k)$. Hence, $(z_i = 1) \iff \bigvee_{j=1}^{n} \psi_{i,j}(Y_1, \dots, Y_k)$. So $\sum_{i=1}^{m} z_i$ is the number of clauses satisfied for any $(Y_1, \dots, Y_k)$ and the optimal solution of the optimization problem gives the minimum number of clauses that remain satisfied for all $(Y_1, \dots, Y_k)$.

13.7 Runtime Analysis of RaVeN

First, we describe the runtime analysis of DiffPoly. Let the original DNN have n neurons. Symbolic bound computations for each variable pair $< x_i^a, x_i^b >$ at worst take $O(n)$ time. Overall, the worst-case complexity for symbolic bound computation for all variable pairs is $O(n^2)$. The *back-substitution* algorithm used for computing concrete bounds in the worst case explores $O(n)$ symbolic bounds before terminating. Obtaining the concrete bounds by substituting concrete values for all variables in each symbolic bound takes $O(n)$ time. The worst-case runtime for obtaining concrete bounds for each variable pair is $O(n^2)$ and the asymptotic runtime of a single DiffPoly analysis is $O(n^3)$. Since we consider $\binom{k}{2}$ pairs of executions the total cost of DiffPoly analysis is $O(k^2 \cdot n^3)$. For product DNN analysis we use an existing DNN abstract interpreter for each of k copies of the original network N. We assume analyzing each copy of N takes C_N time. So analyzing the product DNN takes $k \cdot C_N$ time. For the MILP formulation, we add in the worst-case $O(k)$ of constraints per variable and the product DNN contains $O(k \cdot n)$ variables. Then the total size of the MILP in terms of the number of linear constraints is $O(k^2 \cdot n)$. Since we formulate the MILP using the constraints obtained from the DiffPoly analysis, in the worst case, MILP formulation takes $O(k^2 \cdot n^3)$. Suppose it takes $C_{\mathcal{M}}$ worst case time to optimize the MILP, then worst case time complexity of RaVeN is $O(k^2 \cdot n^3) + k \cdot C_N + C_{\mathcal{M}}$. Note, $C_{\mathcal{M}}$ depends on the MILP encoding of Ψ which is the only source of integer variables in RaVeN's formulation.

14 PROOFS OF LEMMAS

14.1 Proof of lemmas for DiffPoly ReLU transformer

Lemma 14.1. *(Case a in Fig. 4) If $\hat{\delta} = x - y$ where $x, y \in \mathbb{R}$, $\hat{\delta} \in [\hat{\Delta}_{lb}, \hat{\Delta}_{ub}]$ and $\hat{\Delta}_{lb} \geq 0$ then $\delta = ReLU(x) - ReLU(y)$ then $(0 \leq \delta)$ and $(\delta \leq \hat{\delta})$.*

Proof. $\hat{\Delta}_{lb} \geq 0 \implies \hat{\delta} \geq 0 \implies x \geq y$. Now we consider all 3 possible cases below.

$$\text{Case 1} \quad (x \geq 0) \land (y \geq 0) \implies ReLU(x) - ReLU(y) = (x - y) \implies (\delta = \hat{\delta}) \implies (\delta \geq 0)$$

$$\text{Case 2} \quad (x \geq 0) \land (y < 0) \implies ReLU(x) - ReLU(y) = x \implies (\delta \leq (x - y) = \hat{\delta}) \land (\delta \geq 0)$$

$$\text{Case 3} \quad (x < 0) \land (y < 0) \implies ReLU(x) - ReLU(y) = 0 \implies (\delta = 0 \leq \hat{\delta})$$

□

Lemma 14.2. *(Case b in Fig. 4) If $\hat{\delta} = x - y$ where $x, y \in \mathbb{R}$, $\hat{\delta} \in [\hat{\Delta}_{lb}, \hat{\Delta}_{ub}]$ and $\hat{\Delta}_{ub} \leq 0$ then $\delta = ReLU(x) - ReLU(y)$ then $(\hat{\delta} \leq \delta)$ and $(\delta \leq 0)$.*

Proof. $\hat{\Delta}_{ub} \leq 0 \implies \hat{\delta} \leq 0 \implies x \leq y$. Now we consider all 3 possible cases below.

$$\text{Case 1} \quad (x \geq 0) \land (y \geq 0) \implies ReLU(x) - ReLU(y) = (x - y) \implies (\delta = \hat{\delta}) \implies (\delta \leq 0)$$

$$\text{Case 2} \quad (x < 0) \land (y \geq 0) \implies ReLU(x) - ReLU(y) = -y \implies (\delta \geq (x - y) = \hat{\delta}) \land (\delta \leq 0)$$

$$\text{Case 3} \quad (x < 0) \land (y < 0) \implies ReLU(x) - ReLU(y) = 0 \implies (\delta = 0 \geq \hat{\delta})$$

□

Lemma 14.3. *(Case c in Fig. 4) If $\hat{\delta} = x - y$ where $x, y \in \mathbb{R}$, $\hat{\delta} \in [\hat{\Delta}_{lb}, \hat{\Delta}_{ub}]$ and $(\hat{\Delta}_{lb} < 0) \land (\hat{\Delta}_{ub} > 0)$ then $\delta = ReLU(x) - ReLU(y)$ satisfies $(\lambda^{\delta}_{lb} \cdot \hat{\delta} + \mu^{\delta}_{lb} \leq \delta) \land (\delta \leq \lambda^{\delta}_{ub} \cdot \hat{\delta} + \mu^{\delta}_{ub})$ where $\lambda^{\delta}_{ub} = \frac{\hat{\Delta}_{ub}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}}$, $\lambda^{\delta}_{lb} = -\frac{\hat{\Delta}_{lb}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}}$, $-\mu^{\delta}_{ub} = \mu^{\delta}_{lb} = \frac{\hat{\Delta}_{lb} \times \hat{\Delta}_{ub}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}}$.*

Proof. Lemma 14.1 and lemma 14.2 implies $\max(0, \hat{\delta}) \geq \delta$. Next, we show $\lambda^{\delta}_{ub} \cdot \hat{\delta} + \mu^{\delta}_{ub} \geq \max(0, \hat{\delta})$.

$$(\lambda^{\delta}_{ub} > 0) \implies (\forall \hat{\delta} \in [\hat{\Delta}_{lb}, \hat{\Delta}_{ub}]).\ (\lambda^{\delta}_{ub} \cdot \hat{\delta} + \mu^{\delta}_{ub} \geq \frac{\hat{\Delta}_{lb} \times \hat{\Delta}_{ub}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}} + \mu^{\delta}_{ub} = 0)$$

$$(\lambda^{\delta}_{ub} - 1 < 0) \implies (\forall \hat{\delta} \in [\hat{\Delta}_{lb}, \hat{\Delta}_{ub}]).\ (\lambda^{\delta}_{ub} \cdot \hat{\delta} + \mu^{\delta}_{ub} - \hat{\delta} \geq \frac{\hat{\Delta}_{lb} \times \hat{\Delta}_{ub}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}} + \mu^{\delta}_{ub} = 0)$$

$$(\forall \hat{\delta} \in [\hat{\Delta}_{lb}, \hat{\Delta}_{ub}]).\ (\lambda^{\delta}_{ub} \cdot \hat{\delta} + \mu^{\delta}_{ub} \geq \max(0, \hat{\delta})$$

Lemma 14.1 and lemma 14.2 implies $\delta \geq \min(0, \hat{\delta})$. Next, we show $\min(0, \hat{\delta}) \geq \lambda^{\delta}_{lb} \cdot \hat{\delta} + \mu^{\delta}_{lb}$.

$$(\lambda^{\delta}_{lb} > 0) \implies (\forall \hat{\delta} \in [\hat{\Delta}_{lb}, \hat{\Delta}_{ub}]).\ (\lambda^{\delta}_{lb} \cdot \hat{\delta} + \mu^{\delta}_{lb} \leq -\frac{\hat{\Delta}_{lb} \times \hat{\Delta}_{ub}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}} + \mu^{\delta}_{lb} = 0)$$

$$(\lambda^{\delta}_{lb} - 1 < 0) \implies (\forall \hat{\delta} \in [\hat{\Delta}_{lb}, \hat{\Delta}_{ub}]).\ (\lambda^{\delta}_{lb} \cdot \hat{\delta} + \mu^{\delta}_{lb} - \hat{\delta} \leq \frac{\hat{\Delta}_{lb} \times \hat{\Delta}_{ub}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}} + \mu^{\delta}_{lb} = 0)$$

$$(\forall \hat{\delta} \in [\hat{\Delta}_{lb}, \hat{\Delta}_{ub}]).\ (\lambda^{\delta}_{lb} \cdot \hat{\delta} + \mu^{\delta}_{lb} \geq \min(0, \hat{\delta})$$

□

For the cases defined in Table 1, we compute the symbolic bounds $y_i^{a,\leq}$ and $y_i^{a,\geq}$.

$$x_{-}^{a,i} \implies (y_i^{a,\leq} = y_i^{a,\geq} = 0) \quad x_{+}^{a,i} \implies (y_i^{a,\leq} = y_i^{a,\geq} = x_i^a) \quad (x_{\pm}^{a,i}) \land (u_{a,x_i} \geq -l_{a,x_i}) \implies (y_i^{a,\leq} = x_i^a)$$

$$(x_{\pm}^{a,i}) \land (u_{a,x_i} < -l_{a,x_i}) \implies (y_i^{a,\geq} = 0) \qquad (x_{\pm}^{a,i}) \implies (y_i^{a,\geq} = \frac{u_{a,x_i}}{u_{a,x_i} - l_{a,x_i}} \cdot x_i^a - \frac{u_{a,x_i} \times l_{a,x_i}}{u_{a,x_i} - l_{a,x_i}}) \quad (32)$$

Lemma 14.4. *(Correctness of symbolic bounds in Eq. 32) If $x_i^a \in [l_{a,x_i}, u_{a,x_i}]$ then $y_i^a = ReLU(x_i^a)$ then $y_i^{a,\leq} \leq y_i^a \leq y_i^{a,\geq}$ where $y_i^{a,\leq}$ and $y_i^{a,\geq}$ defined in Eq. 32.*

Proof. Refer to proof of Theorem 4.2 of [69]. □

Lemma 4.2. *(Correctness of symbolic bounds in Table 2 and 3) If $x_i^a \in [l_{a,x_i}, u_{a,x_i}]$, $x_i^b \in [l_{b,x_i}, u_{b,x_i}]$ and $\delta_{x_i}^{a,b} = (x_i^a - x_i^b) \in [\Delta_{lb}^{a,b,x_i}, \Delta_{ub}^{a,b,x_i}]$ and $\delta_{y_i}^{a,b} = ReLU(x_i^a) - ReLU(x_i^b)$ then $\delta_{y_i}^{a,b,\leq} \leq \delta_{y_i}^{a,b} \leq \delta_{y_i}^{a,b,\geq}$ where $\delta_{y_i}^{a,b,\leq}$ and $\delta_{y_i}^{a,b,\geq}$ defined in Table 2 and 3.*

Proof. We show in all 12 cases shown in Table 2 and Table 3 $\delta_{y_i}^{a,b,\leq} \leq \delta_{y_i}^{a,b} \leq \delta_{y_i}^{a,b,\geq}$ holds.

- Case 1: $x_-^{a,i} \wedge x_-^{b,i} \implies (ReLU(x_i^a = 0) \wedge (ReLU(x_i^b = 0)) \implies \delta_{y_i}^{a,b} = 0$
- Case 2: $x_+^{a,i} \wedge x_+^{b,i} \implies (ReLU(x_i^a) = x_i^a) \wedge (ReLU(x_i^b) = x_i^b) \implies \delta_{y_i}^{a,b} = x_i^a - x_i^b = \delta_{x_i}^{a,b}$.
- Case 3: $x_+^{a,i} \wedge x_-^{b,i} \implies (ReLU(x_i^a) = x_i^a) \wedge (ReLU(x_i^b) = 0) \implies \delta_{y_i}^{a,b} = x_i^a$.
- Case 4: $x_-^{a,i} \wedge x_+^{b,i} \implies (ReLU(x_i^a) = 0) \wedge (ReLU(x_i^b) = x_i^b) \implies \delta_{y_i}^{a,b} = -x_i^b$.
- Case 5: $x_\pm^{a,i} \wedge x_-^{b,i} \implies (ReLU(x_i^b) = 0) \implies \delta_{y_i}^{a,b} = y_i^a \implies y_i^{a,\leq} \leq \delta_{y_i}^{a,b} \leq y_i^{a,\geq}$.
- Case 6: $x_-^{a,i} \wedge x_\pm^{b,i} \implies (ReLU(x_i^a) = 0) \implies \delta_{y_i}^{a,b} = -y_i^b \implies -y_i^{b,\geq} \leq \delta_{y_i}^{a,b} \leq -y_i^{b,\leq}$.
- Case 7: $x_\pm^{a,i} \wedge x_+^{b,i} \implies (ReLU(x_i^b) = x_i^b) \implies \delta_{y_i}^{a,b} = y_i^a - x_i^b \implies y_i^{a,\leq} - x_i^b \leq \delta_{y_i}^{a,b} \leq y_i^{a,\geq} - x_i^b$.
- Case 8: $x_+^{a,i} \wedge x_\pm^{b,i} \implies (ReLU(x_i^a) = x_i^a) \implies \delta_{y_i}^{a,b} = x_i^a - y_i^b \implies x_i^a - y_i^{b,\geq} \leq \delta_{y_i}^{a,b} \leq x_i^a - y_i^{b,\leq}$.
- Case 9: $x_\pm^{a,i} \wedge x_\pm^{b,i} \implies \delta_{y_i}^{a,b} = y_i^a - y_i^b \implies y_i^{a,\leq} - y_i^{b,\geq} \leq \delta_{y_i}^{a,b} \leq y_i^{a,\geq} - y_i^{b,\leq}$.
- Case 10: $\delta_+ \implies 0 \leq \delta_{y_i}^{a,b} \leq \delta_{x_i}^{a,b}$ from Lemma 14.1.
- Case 11: $\delta_- \implies \delta_{x_i}^{a,b} \leq \delta_{y_i}^{a,b} \leq 0$ from Lemma 14.2.
- Case 12: $\delta_\pm \implies \lambda_{lb}^{\delta} \delta_{x_i}^{a,b} + \mu_{lb}^{\delta} \leq \delta_{y_i}^{a,b} \leq \lambda_{ub}^{\delta} \delta_{x_i}^{a,b} + \mu_{ub}^{\delta}$ from Lemma 14.3.

□

Lemma 4.3. *(Correctness of concrete bounds computed by the ReLU transformer) If $x_i^a \in [l_{a,x_i}, u_{a,x_i}]$, $x_i^b \in [l_{b,x_i}, u_{b,x_i}]$ and $\delta_{x_i}^{a,b} = (x_i^a - x_i^b) \in [\Delta_{lb}^{a,b,x_i}, \Delta_{ub}^{a,b,x_i}]$, $y_i^a = ReLU(x_i^a)$, $y_i^b = ReLU(x_i^b)$, $\delta_{y_i}^{a,b} = y_i^a - y_i^b$ then $l_{a,y_i} \leq y_i^a \leq u_{a,y_i}$, $l_{b,y_i} \leq y_i^b \leq u_{b,y_i}$, and $\Delta_{lb}^{a,b,y_i} \leq \delta_{y_i}^{a,b} \leq \Delta_{ub}^{a,b,y_i}$ where Δ_{lb}^{a,b,y_i} and Δ_{ub}^{a,b,y_i} computed by applying back-substitution on $\delta_{y_i}^{a,b,\leq}$ and $\delta_{y_i}^{a,b,\geq}$ respectively.*

Proof. The concrete bounds $l_{a,y_i}, l_{b,y_i}, u_{a,y_i}, u_{b,y_i}$ are obtained from the analysis of product DNN with existing DNN abstract interpreter. The existing DNN abstract interpreter ensures the concrete lower and upper bounds always satisfy the following - $l_{a,y_i} \leq y_i^a \leq u_{a,y_i}$, $l_{b,y_i} \leq y_i^b \leq u_{b,y_i}$. Now, the concrete bounds Δ_{lb}^{a,b,y_i} and Δ_{ub}^{a,b,y_i} are obtained with back-substitution starting with symbolic bounds $\delta_{y_i}^{a,b,\leq}$ and $\delta_{y_i}^{a,b,\geq}$ respectively. From Lemma 4.2 we show that $(\delta_{y_i}^{a,b,\leq} \leq \delta_{y_i}^{a,b}) \wedge (\delta_{y_i}^{a,b} \leq \delta_{y_i}^{a,b,\geq})$ holds. Since, $(\delta_{y_i}^{a,b,\leq} \leq \delta_{y_i}^{a,b}) \wedge (\delta_{y_i}^{a,b} \leq \delta_{y_i}^{a,b,\geq})$ using Lemma 12.1 we show that $\Delta_{lb}^{a,b,y_i} \leq \delta_{y_i}^{a,b}$ and $\delta_{y_i}^{a,b} \leq \Delta_{ub}^{a,b,y_i}$. □

14.2 Proof of lemmas for DiffPoly Sigmoid and Tanh transformer

For the rest of this section, we assume the function $g : \mathbb{R} \to \mathbb{R}$ is differentiable everywhere. We use $l_{g'}$ and $u_{g'}$ to denote minimum and maximum value of g' (derivative of g) for the range $[l, u]$ where $l = \min(l_{a,x_i}, l_{b,x_i})$ and $u = \max(u_{a,x_i}, u_{b,x_i})$. Here, $l_{g'} = \min_{x \in [l,u]} g'(x)$ and $u_{g'} = \max_{x \in [l,u]} g'(x)$

Lemma 14.5. *If $\hat{\delta} = x - y$ where $x, y \in \mathbb{R}$, $\hat{\delta} \in [\hat{\Delta}_{lb}, \hat{\Delta}_{ub}]$, $x \in [l, u]$, $y \in [l, u]$ and $\hat{\Delta}_{lb} \geq 0$ then $\delta = g(x) - g(y)$ then $(l_{g'} \cdot \hat{\delta} \leq \delta)$ and $(\delta \leq u_{g'} \cdot \hat{\delta})$.*

Proof. Since g is differentiable everywhere by using the Mean Value Theorem

$$\begin{aligned} \frac{g(x)-g(y)}{x-y} &= f'(c) \text{ where } c \in [l,u] \\ l_{g'} \leq \frac{g(x)-g(y)}{x-y} &\leq u_{g'} \end{aligned} \tag{33}$$

Now $\hat{\Delta}_{lb} \geq 0 \implies \hat{\delta} \geq 0 \implies (x-y) \geq 0$.

$$\begin{aligned} (x-y) \geq 0 &\implies (l_{g'} \cdot (x-y) \leq (g(x)-g(y)) \text{ using Eq. 33} \\ (x-y) \geq 0 &\implies ((g(x)-g(y) \leq u_{g'} \cdot (x-y)) \text{ using Eq. 33} \end{aligned}$$

□

Lemma 14.6. *If $\hat{\delta} = x-y$ where $x, y \in \mathbb{R}$, $\hat{\delta} \in [\hat{\Delta}_{lb}, \hat{\Delta}_{ub}]$, $x \in [l,u]$, $y \in [l,u]$ and $\hat{\Delta}_{ub} \leq 0$ then $\delta = g(x)-g(y)$ then $(u_{g'} \cdot \hat{\delta} \leq \delta)$ and $(\delta \leq l_{g'} \cdot \hat{\delta})$.*

Proof. Now $\hat{\Delta}_{ub} \leq 0 \implies \hat{\delta} \leq 0 \implies (x-y) \leq 0$.

$$\begin{aligned} (x-y) \leq 0 &\implies (u_{g'} \cdot (x-y) \leq (g(x)-g(y)) \text{ using Eq. 33} \\ (x-y) \leq 0 &\implies ((g(x)-g(y) \leq l_{g'} \cdot (x-y)) \text{ using Eq. 33} \end{aligned}$$

□

Lemma 14.7. *If $\hat{\delta} = x-y$ where $x, y \in \mathbb{R}$, $\hat{\delta} \in [\hat{\Delta}_{lb}, \hat{\Delta}_{ub}]$ and $(\hat{\Delta}_{lb} < 0)$ and $(\hat{\Delta}_{ub} > 0)$ then $\delta = g(x)-g(y)$ satisfies $(\lambda^{\delta}_{lb} \cdot \hat{\delta} + \mu^{\delta}_{lb} \leq \delta)$ and $(\delta \leq \lambda^{\delta}_{ub} \cdot \hat{\delta} + \mu^{\delta}_{ub})$ where $\lambda^{\delta}_{ub} = \frac{u_{g'} \times \hat{\Delta}_{ub} - l_{g'} \times \hat{\Delta}_{lb}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}}$, $\lambda^{\delta}_{lb} = \frac{l_{g'} \times \hat{\Delta}_{ub} - u_{g'} \times \hat{\Delta}_{lb}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}}$, $-\mu^{\delta}_{ub} = \mu^{\delta}_{lb} = \frac{(u_{g'} - l_{g'}) \times \hat{\Delta}_{lb} \times \hat{\Delta}_{ub}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}}$.*

Proof. Lemma 14.5 and lemma 14.6 implies $\max(l_{g'} \cdot \hat{\delta}, u_{g'} \cdot \hat{\delta}) \geq \delta$. Next, we show $\lambda^{\delta}_{ub} \cdot \hat{\delta} + \mu^{\delta}_{ub} \geq \max(l_{g'} \cdot \hat{\delta}, u_{g'} \cdot \hat{\delta})$.

$$\begin{aligned} (\lambda^{\delta}_{ub} - l_{g'}) \cdot \hat{\delta} = \frac{(u_{g'} - l_{g'}) \times \hat{\Delta}_{ub}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}} \cdot \hat{\delta} &\geq \frac{(u_{g'} - l_{g'}) \times \hat{\Delta}_{ub} \times \hat{\Delta}_{lb}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}} \\ \implies (\lambda^{\delta}_{ub} - l_{g'}) \cdot \hat{\delta} + \mu^{\delta}_{ub} &\geq \frac{(u_{g'} - l_{g'}) \times \hat{\Delta}_{ub} \times \hat{\Delta}_{lb}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}} + \mu^{\delta}_{ub} = 0 \\ \implies \lambda^{\delta}_{ub} \cdot \hat{\delta} + +\mu^{\delta}_{ub} &\geq l_{g'} \cdot \hat{\delta} \end{aligned} \tag{34}$$

$$\begin{aligned} (\lambda^{\delta}_{ub} - u_{g'}) \cdot \hat{\delta} = \frac{(u_{g'} - l_{g'}) \times \hat{\Delta}_{lb}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}} \cdot \hat{\delta} &\geq \frac{(u_{g'} - l_{g'}) \times \hat{\Delta}_{ub} \times \hat{\Delta}_{lb}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}} \\ \implies (\lambda^{\delta}_{ub} - u_{g'}) \cdot \hat{\delta} + \mu^{\delta}_{ub} &\geq \frac{(u_{g'} - l_{g'}) \times \hat{\Delta}_{ub} \times \hat{\Delta}_{lb}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}} + \mu^{\delta}_{ub} = 0 \\ \implies \lambda^{\delta}_{ub} \cdot \hat{\delta} + +\mu^{\delta}_{ub} &\geq u_{g'} \cdot \hat{\delta} \end{aligned} \tag{35}$$

Combining results from Eq. 34 and Eq. 35 we show that $\lambda^{\delta}_{ub} \cdot \hat{\delta} + \mu^{\delta}_{ub} \geq \max(l_{g'} \cdot \hat{\delta}, u_{g'} \cdot \hat{\delta}) \geq \delta$.
Lemma 14.5 and lemma 14.6 implies $\delta \geq \min(l_{g'} \cdot \hat{\delta}, u_{g'} \cdot \hat{\delta})$.
Next, we show $\min(l_{g'} \cdot \hat{\delta}, u_{g'} \cdot \hat{\delta}) \geq \lambda^{\delta}_{lb} \cdot \hat{\delta} + \mu^{\delta}_{lb}$.

$$\begin{aligned}
(l_{g'} - \lambda^{\delta}_{ub}) \cdot \hat{\delta} &= \frac{(u_{g'} - l_{g'}) \times \hat{\Delta}_{lb}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}} \cdot \hat{\delta} \geq \frac{(u_{g'} - l_{g'}) \times \hat{\Delta}_{ub} \times \hat{\Delta}_{lb}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}} \\
\implies (l_{g'} - \lambda^{\delta}_{ub}) \cdot \hat{\delta} - \mu^{\delta}_{lb} &\geq \frac{(u_{g'} - l_{g'}) \times \hat{\Delta}_{ub} \times \hat{\Delta}_{lb}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}} - \mu^{\delta}_{lb} = 0 \\
\implies l_{g'} \cdot \hat{\delta} &\geq \lambda^{\delta}_{lb} \cdot \hat{\delta} + +\mu^{\delta}_{lb} \qquad (36)
\end{aligned}$$

$$\begin{aligned}
(u_{g'} - \lambda^{\delta}_{ub}) \cdot \hat{\delta} &= \frac{(u_{g'} - l_{g'}) \times \hat{\Delta}_{ub}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}} \cdot \hat{\delta} \geq \frac{(u_{g'} - l_{g'}) \times \hat{\Delta}_{ub} \times \hat{\Delta}_{lb}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}} \\
\implies (u_{g'} - \lambda^{\delta}_{ub}) \cdot \hat{\delta} - \mu^{\delta}_{lb} &\geq \frac{(u_{g'} - l_{g'}) \times \hat{\Delta}_{ub} \times \hat{\Delta}_{lb}}{\hat{\Delta}_{ub} - \hat{\Delta}_{lb}} - \mu^{\delta}_{lb} = 0 \\
\implies u_{g'} \cdot \hat{\delta} &\geq \lambda^{\delta}_{lb} \cdot \hat{\delta} + +\mu^{\delta}_{lb} \qquad (37)
\end{aligned}$$

Combining results from Eq. 36 and Eq. 37 we show that $\lambda^{\delta}_{lb} \cdot \hat{\delta} + \mu^{\delta}_{lb} \leq \min(l_{g'} \cdot \hat{\delta}, u_{g'} \cdot \hat{\delta}) \leq \delta$. □

14.3 Proof of soundness for DiffPoly Affine transformer

Lemma 14.8. *For $y \leftarrow v + \sum_{i=1}^{n} w_i \cdot x_i$ and $\forall i \in [n].(x_i^{\leq} \leq x_i) \wedge (x_i \leq x_i^{\geq})$ then $y \leq v + \sum_{i=1}^{n} w_i^{+} \cdot x_i^{\geq} + \sum_{i=1}^{n} w_i^{-} \cdot x_i^{\leq}$ where $v, w_1, \dots w_n \in \mathbb{R}$ and $w_i^{-} = \min(w_i, 0)$ and $w_i^{+} = \max(w_i, 0)$.*

Proof. $w_i^{-} \leq 0 \implies w_i^{-} \cdot x_i \leq w_i^{-} \cdot x_i^{\leq}$ and $w_i^{+} \geq 0 \implies w_i^{+} \cdot x_i \leq w_i^{+} \cdot x_i^{\geq}$. Since $(\forall i \in [n]).(w_i^{-} \cdot x_i + w_i^{+} \cdot x_i = w_i \cdot x_i)$ then

$$y = v + \sum_{i=1}^{n} w_i \cdot x_i = v + \sum_{i=1}^{n} w_i^{-} \cdot x_i + w_i^{+} \cdot x_i \leq v + \sum_{i=1}^{n} w_i^{+} \cdot x_i^{\geq} + \sum_{i=1}^{n} w_i^{-} \cdot x_i^{\leq}$$

□

Lemma 14.9. *For $y \leftarrow v + \sum_{i=1}^{n} w_i \cdot x_i$ and $\forall i \in [n].(x_i^{\leq} \leq x_i) \wedge (x_i \leq x_i^{\geq})$ then $y \geq v + \sum_{i=1}^{n} w_i^{+} \cdot x_i^{\leq} + \sum_{i=1}^{n} w_i^{-} \cdot x_i^{\geq}$ where $v, w_1, \dots w_n \in \mathbb{R}$ and $w_i^{-} = \min(w_i, 0)$ and $w_i^{+} = \max(w_i, 0)$.*

Proof. $w_i^{-} \leq 0 \implies w_i^{-} \cdot x_i \geq w_i^{-} \cdot x_i^{\geq}$ and $w_i^{+} \geq 0 \implies w_i^{+} \cdot x_i \geq w_i^{+} \cdot x_i^{\leq}$. Since $(\forall i \in [n]).(w_i^{-} \cdot x_i + w_i^{+} \cdot x_i = w_i \cdot x_i)$ then

$$y = v + \sum_{i=1}^{n} w_i \cdot x_i = v + \sum_{i=1}^{n} w_i^{-} \cdot x_i + w_i^{+} \cdot x_i \geq v + \sum_{i=1}^{n} w_i^{-} \cdot x_i^{\geq} + \sum_{i=1}^{n} w_i^{+} \cdot x_i^{\leq}$$

□

Lemma 13.4. *(Correctness of symbolic bounds computed by the affine transformer) If $\forall j \in [i].\ x^a_j \in [l_{a,x_j}, u_{a,x_j}]$, $\forall j \in [i].\ x^b_j \in [l_{b,x_j}, u_{b,x_j}]$ and $\forall j \in [i].\ \delta^{a,b}_{x_j} \in [\Delta^{a,b,x_j}_{lb}, \Delta^{a,b,x_j}_{ub}]$ and $x^a_{i+1} = v + \sum_{j=1}^{i} w_j \cdot x^a_j$, $x^b_{i+1} = v + \sum_{j=1}^{i} w_j \cdot x^b_j$, and $\delta^{a,b}_{x_{i+1}} = (x^a_{i+1} - x^b_{i+1})$ then $x^{a,\leq}_{i+1} \leq x^a_{i+1} \leq x^{a,\geq}_{i+1}$, $x^{b,\leq}_{i+1} \leq x^b_{i+1} \leq x^{b,\geq}_{i+1}$ and $\delta^{a,b,\leq}_{x_{i+1}} \leq \delta^{a,b}_{x_{i+1}} \leq \delta^{a,b,\geq}_{x_{i+1}}$ where $x^{a,\leq}_{i+1}, x^{a,\geq}_{i+1}, x^{b,\leq}_{i+1}, x^{b,\geq}_{i+1}, \delta^{a,b,\leq}_{x_{i+1}}$ and $\delta^{a,b,\geq}_{x_{i+1}}$ defined in Eq. 8.*

Proof. We use the results of Lemma 14.8 and Lemma 14.8 to show the correctness of the symbolic bounds.

$$\begin{aligned}
&(x^a_{i+1} \leq x^{a,\geq}_{i+1}) \wedge (x^b_{i+1} \leq x^{b,\geq}_{i+1}) \wedge (\delta^{a,b}_{x_{i+1}} \leq \delta^{a,b,\geq}_{x_{i+1}}) && \text{From lemma 14.8} \\
&(x^a_{i+1} \geq x^{a,\leq}_{i+1}) \wedge (x^b_{i+1} \geq x^{b,\leq}_{i+1}) \wedge (\delta^{a,b}_{x_{i+1}} \geq \delta^{a,b,\leq}_{x_{i+1}}) && \text{From lemma 14.9}
\end{aligned}$$

□

Lemma 13.5. *(Correctness of concrete bounds computed by the affine transformer) If* $\forall j \in [i].\ x^a_j \in [l_{a,x_j}, u_{a,x_j}]$, $\forall j \in [i].\ x^b_j \in [l_{b,x_j}, u_{b,x_j}]$ *and* $\forall j \in [i].\ \delta^{a,b}_{x_j} \in [\Delta^{a,b,x_j}_{lb}, \Delta^{a,b,x_j}_{ub}]$ *and* $x^a_{i+1} = v + \sum_{j=1}^{i} w_j \cdot x^a_j$, $x^b_{i+1} = v + \sum_{j=1}^{i} w_j \cdot x^b_j$, *and* $\delta^{a,b}_{x_{i+1}} = (x^a_{i+1} - x^b_{i+1})$ *then* $l_{a,x_{i+1}} \leq x^a_{i+1} \leq u_{a,x_{i+1}}$, $l_{b,x_{i+1}} \leq x^b_{i+1} \leq u_{b,x_{i+1}}$ *and* $\Delta^{a,b,x_{i+1}}_{lb} \leq \delta^{a,b}_{x_{i+1}} \leq \Delta^{a,b,x_{i+1}}_{ub}$.

Proof. The concrete bounds $l_{a,x_{i+1}}, l_{b,x_{i+1}}, u_{a,x_{i+1}}, u_{b,x_{i+1}}$ are obtained from the analysis of product DNN with existing DNN abstract interpreter. The existing DNN abstract interpreter ensures the concrete lower and upper bounds always satisfy the following - $l_{a,x_{i+1}} \leq x^a_{i+1} \leq u_{a,x_{i+1}}$, $l_{b,x_{i+1}} \leq x^b_{i+1} \leq u_{b,x_{i+1}}$. Now, the concrete bounds $\Delta^{a,b,x_{i+1}}_{lb}$ and $\Delta^{a,b,x_{i+1}}_{ub}$ are obtained with back-substitution starting with symbolic bounds $\delta^{a,b,\leq}_{x_{i+1}}$ and $\delta^{a,b,\geq}_{x_{i+1}}$ respectively. From Lemma 13.4 we show that $(\delta^{a,b,\leq}_{x_{i+1}} \leq \delta^{a,b}_{x_{i+1}}) \wedge (\delta^{a,b}_{x_{i+1}} \leq \delta^{a,b,\geq}_{x_{i+1}})$ holds. Since, $(\delta^{a,b,\leq}_{x_{i+1}} \leq \delta^{a,b}_{x_{i+1}}) \wedge (\delta^{a,b}_{x_{i+1}} \leq \delta^{a,b,\geq}_{x_{i+1}})$ using Lemma 12.1 we show that $\Delta^{a,b,x_{i+1}}_{lb} \leq \delta^{a,b}_{x_{i+1}}$ and $\delta^{a,b}_{x_{i+1}} \leq \Delta^{a,b,x_{i+1}}_{ub}$. □

14.4 Specific MILP encoding UAP, hamming distance and targeted UAP

UAP MILP objective encoding

$$\min_{(Y_1,\dots,Y_k)} \sum_{i=1}^{k} z_i \quad \text{s.t.}$$

$$x_{i,j} = \psi_{i,j}(Y_1,\dots,Y_k) = \left(C_{i,j}^T Y_i \geq 0\right) \quad j \in [n_l] \text{ and } C_{i,j} \text{ from Eq. 12}$$

$$z_i = \left(\sum_{j=1}^{n_l} x_{i,j} \geq n_l\right) \quad i \in [k]$$

Hamming distance MILP objective encoding

$$\max_{(Y_1,\dots,Y_k)} k - \sum_{i=1}^{k} z_i \quad \text{s.t.}$$

$$x_{i,j} = \psi_{i,j}(Y_1,\dots,Y_k) = \left(C_{i,j}^T Y_i \geq 0\right) \quad j \in [n_l] \text{ and } C_{i,j} \text{ from Eq. 12}$$

$$z_i = \left(\sum_{j=1}^{n_l} x_{i,j} \geq n_l\right) \quad i \in [k]$$

Targeted UAP MILP objective encoding

$$\min_{(Y_1,\dots,Y_k)} \sum_{i=1}^{k} z_i \quad \text{s.t.}$$

$$x_{i,j} = \psi_{i,j}(Y_1,\dots,Y_k) = \left(C_{i,j}^T Y_i \geq 0\right) \quad j \in [n_l] \text{ and } C_{i,j} \text{ from Eq. 13}$$

$$z_i = \left(\sum_{j=1}^{n_l} x_{i,j} \geq 0\right) \quad i \in [k]$$

14.5 Generalization of DiffPoly

In this section, we discuss how DiffPoly can be generalized for computing bounds on any general linear combination specified by of the layerwise outputs of any k DNN executions. This will enable us to handle relational properties where the cross-execution input constraint bounds a general linear combination of inputs used in different executions rather than bounding pairwise input differences. However, to the best of our knowledge, for most of the common DNN relational properties, the cross-execution input constraints are limited to bounding differences. For k executions, the general form of cross-execution input constraint is as follows where $X_1, \dots, X_k \in \mathbb{R}^{n_0}$ are inputs to k executions and $a_1, \cdots, a_k \in \mathbb{R}$ are constant real numbers and $L \in \mathbb{R}^{n_0}$ and U are constant vectors:

$$L \leq \sum_{i=1}^{k} a_i \cdot X_i \leq U \tag{38}$$

We consider k copies of the same variable $< x_i^1, \dots, x_i^k >$ one from from each of k executions and use δ_i^x to denote linear combination of all x_i^j where $j \in [k]$ i.e. $\delta_i^x = \sum_{j=1}^{k} a_j \cdot x_i^j$. Now, similar to DiffPoly, we discuss how we handle affine and activation assignments involving the variables $< x_1^1, \dots, x_1^k > \dots < x_n^1, \dots, x_n^k >$ and compute symbolic and concrete bounds on $\delta_i^x = \sum_{j=1}^{k} a_j \cdot x_i^j$ for each variable in N where a_js are fixed reals. The symbolic bounds follow the same format as de

Affine assignments: We consider the following k affine assignments.

$$x_{n+1}^1 \leftarrow \sum_{i=1}^{n} w_i \cdot x_i^1 + b \qquad x_{n+1}^2 \leftarrow \sum_{i=1}^{n} w_i \cdot x_i^2 + b$$

$$\dots \qquad \dots$$

$$x_{n+1}^{k-1} \leftarrow \sum_{i=1}^{n} w_i \cdot x_i^{k-1} + b \qquad x_{n+1}^k \leftarrow \sum_{i=1}^{n} w_i \cdot x_i^k + b$$

Then if $\delta_{n+1}^x = \sum_{j=1}^{k} a_j \cdot x_{n+1}^j$ then $\delta_{n+1}^x = \sum_{i=1}^{n} w_i \cdot \delta_i^x + b \cdot \sum_{i=1}^{k} a_i$. Given, δ_{n+1}^x is already a linear function of δ_j^x where $j \in n$, the symbolic bounds δ_{n+1}^x can directly computed as shown below

$$\delta_{n+1}^{x,\leq} = \delta_{n+1}^{x,\geq} = \sum_{i=1}^{n} w_i \cdot \delta_i^x + b \cdot \sum_{i=1}^{k} a_i$$

The concrete bounds of δ_{n+1}^x in this case are obtained by back substitution.

Non-linear activation assignments: We consider the following k assignments involving a non-linear activation $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ like ReLU, Sigmoid, Tanh, etc.

$$y_n^1 \leftarrow \sigma(x_n^1) \qquad y_n^2 \leftarrow \sigma(x_n^2)$$

$$\dots \qquad \dots$$

$$y_n^{k-1} \leftarrow \sigma(x_n^{k-1}) \qquad y_n^k \leftarrow \sigma(x_n^k)$$

Let, $l = \min_{i \in [k]} l_n^i$ and $u = \max_{i \in [k]} u_n^i$ where for all $i \in [k]$ $l_n^i \leq x_n^i \leq u_n^i$. Next, we use the linear overapproximation of popular activation functions including ReLU, Sigmoid and Tanh used in DeepZ [68] utilizing the bounds l, u. Given, l and u DeepZ computes linear bounds specified by λ_σ, μ such that $\mu \geq 0$for all $x \in [l, u]$ following inequalities holds:

$$\lambda_\sigma \cdot x - \mu \leq \sigma(x) \leq \lambda_\sigma \cdot x + \mu$$

Now we will compute the symbolic bounds for $\delta_n^y = \sum_{i=1}^{k} a_i \cdot y_n^i$. For all $x \in [l, u]$ and real number $a \in \mathbb{R}$ following inequality holds

$$a \cdot \lambda_\sigma \cdot x - |a| \cdot \mu \leq a \cdot \sigma(x) \leq a \cdot \lambda_\sigma \cdot x + |a| \cdot \mu$$

Given for all $i \in [k]$ $l \leq x_n^i \leq u$, then

$$a_i \cdot \lambda_\sigma \cdot x^i - |a_i| \cdot \mu \leq a_i \cdot \sigma(x_i^n) \leq a_i \cdot \lambda_\sigma \cdot x + |a_i| \cdot \mu \quad \forall i \in [k]$$

Symbolic bounds of δ_n^y are as follows:

$$\left(\sum_{i=1}^{k} a_i \cdot \lambda_\sigma \cdot x_n^i - |a_i|\right) \cdot \mu) \leq \sum_{i=1}^{k} a_i \cdot \sigma(x_i^n) \leq \left(\sum_{i=1}^{k} a_i \cdot \lambda_\sigma \cdot x + |a_i| \cdot \mu\right)$$

$$\lambda_\sigma \delta^x n - \mu \cdot \sum_{i=1}^{k} |a_i| \leq \delta_n^y \leq \lambda_\sigma \delta^x n + \mu \cdot \sum_{i=1}^{k} |a_i|$$

The concrete bounds of δ_{n+1}^x in this case are obtained by back substitution.

15 ADDITIONAL EXPERIMENTS

15.1 Targeted UAP Verification

In this section, we show results for the targeted UAP verification problem. We see that RaVeN outperforms both baselines significantly. Figure 15 shows RaVeN and baseline approaches performance on each class with a standardly trained ConvSmall network on CIFAR10 with $\epsilon = 4/255$. For example, when targeting the 8th label we see that RaVeN achieves an average worst-case accuracy of 70% compared to 33% achieved by the two baselines.

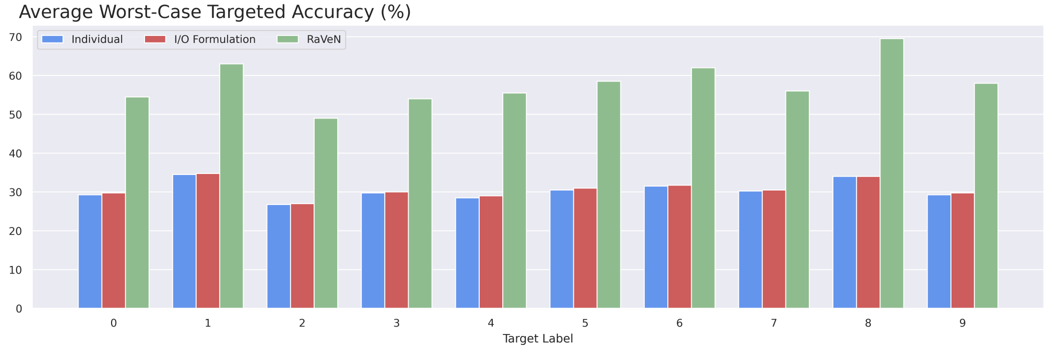
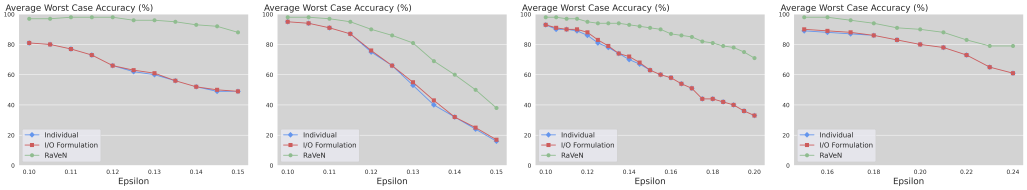


Fig. 15. Average Worst case targeted UAP accuracy over all classes for ConvSmall on CIFAR10 with $\epsilon = 4/255$

15.2 Ablation on using different Individual Verifiers

In this section, we show results using DeepPoly [69] instead of DeepZ [68]. Similarly to when using DeepZ we see that RaVeN obtains better performance when compared the the baselines for all networks and ϵs.



(a) IBP-Small (MNIST) (b) ConvSmall (MNIST) (c) IBP (MNIST) (d) ConvBig (MNIST)

Fig. 16. RaVeN results with DeepPoly as the baseline verifier.

15.3 RaVeN Layerwise Formulation Runtimes

In Table 7, we show the runtime comparision of RaVeN Layerwise (LW) formulation and RaVeN with difference constraints on networks shown in Figure 11. We note that the primary increase in computation time we observe comes from running DiffPoly. For networks which incur additional cost in MILP time with difference constraints (RaVeN MILP Time vs Layerwise MILP Time) we believe that the increase in performance justifies this cost. For example, for hamming distance verification, RaVeN Layerwise does not improve over the two baseline approaches. Only by adding the difference constraints do we see a performance jump over the baselines.

Table 7. Runtime Comparison of RaVeN Layerwise formulation and RaVeN with difference constraints

Dataset	Model	Ind. Veri.	I/O Form.	RaVeN	RaVeN LW	RaVeN MILP Time	LW MILP Time
MNIST	IBP-Small	0.04	0.12	1.98	1.01	1.06	0.96
MNIST	ConvSmall	0.30	0.38	7.40	4.98	4.06	4.66
CIFAR10	IBP-Small	0.29	0.47	8.39	3.94	5.03	3.63
MNIST	Hamming (Sigmoid)	0.03	0.13	1.41	0.46	1.34	0.45