End-To-End Set-Based Training for Neural Network Verification

Lukas Koller
lukas.koller@tum.de
Technical University of Munich
Munich, Germany

Tobias Ladner
tobias.ladner@tum.de
Technical University of Munich
Munich, Germany

Matthias Althoff
althoff@tum.de
Technical University of Munich
Munich, Germany

ABSTRACT

Neural networks are vulnerable to adversarial attacks, i.e., small input perturbations can result in substantially different outputs of a neural network. Safety-critical environments require neural networks that are robust against input perturbations. However, training and formally verifying robust neural networks is challenging. We address this challenge by employing, for the first time, a end-to-end set-based training procedure that trains robust neural networks for formal verification. Our training procedure drastically simplifies the subsequent formal robustness verification of the trained neural network. While previous research has predominantly focused on augmenting neural network training with adversarial attacks, our approach leverages set-based computing to train neural networks with entire sets of perturbed inputs. Moreover, we demonstrate that our set-based training procedure effectively trains robust neural networks, which are easier to verify. In many cases, set-based trained neural networks outperform neural networks trained with state-of-the-art adversarial attacks.

KEYWORDS

Neural network verification, formal verification, set-based computing, zonotopes.

1 INTRODUCTION

Neural networks achieve impressive results for many complex tasks, such as speech recognition [17] or object detection [39]. However, many neural networks are sensitive to input perturbations [37]: Small, carefully chosen input changes can lead to vastly different outputs. This behavior is problematic for the safe adoption of neural networks in safety-critical scenarios, e.g., autonomous vehicle control [43], airborne collision avoidance [18], or operation of nuclear reactors [6]. Thus, the formal verification of the robustness of neural networks gained interest in recent years [5, 11]. A neural network is considered robust against a perturbance set if it returns the correct output for every value within the perturbance set, typically an l_∞-ball. Subsequently, we provide a brief overview of related work.

1.1 Related Work

Adversarial Attacks. An adversarial attack is a modified input within a perturbance set that leads to an incorrect output of a neural network. The most prominent approaches to generating adversarial attacks are the fast gradient sign method (FGSM) and projected gradient descent (PGD). FGSM is a single-step gradient-based adversarial attack that efficiently generates adversarial attacks [15]. PGD uses multiple iterations of FGSM to compute stronger adversarial attacks [22].

Training Robust Neural Networks. The training objective of a robust neural network is typically formulated as a min-max optimization problem [27]: Minimize the worst-case input within a perturbance set. A worst-case input maximizes the error of the corresponding output with respect to a target output. Computing a worst-case input within a perturbance set is computationally difficult [40]. Nonetheless, neural networks are effectively trained by approximating worst-case inputs with adversarial attacks, e.g., computed with PGD [27]. Other approaches for training robust neural networks are via input gradient regularization [33] or neural network destillation [32]. However, none of these training approaches incorporate formal verification methods.

Formal Robustness Verification of Neural Networks. The formal robustness verification of neural networks is crucial to safely deploy neural networks in safety-critical scenarios. Most formal verification approaches either formulate the verification problem as an optimization problem or use reachability analysis [5]. Optimization-based approaches encode the verification problem as an optimization problem, which is solved using (mixed-integer) linear programming [29, 45, 36] or satisfiability modulo theories (SMT) [19] solvers. Alternatively, reachability analysis uses efficient set representations, e.g., zonotopes [13], combined with set-based computations to enclose the output set of a neural network for a given input set [21, 23, 12, 35]. The output set of a neural network is used to verify its safety.

Combined Training and Robustness Verification of Neural Networks. Many approaches combine the training and formal verification of robust neural networks by enhancing the training using adversarial attacks. In these works, the approximation of a worst-case input is replaced by an upper bound, which can verify that no perturbation will lead to an incorrect output. Different methods for upper-bounding a worst-case input have been proposed: Interval bound propagation (IBP) [16, 26], linear relaxation [44], (mixed-integer) linear programming [41], or abstract interpretation [28]. Some approaches use set-based methods to compute upper bounds [28]; however, only the upper bound is used for training, discarding much set-based information. Conversely, our approach is end-to-end set-based, meaning we use the entire sets for training. Moreover, training neural networks with adversarial attacks or upper bounds reduces their accuracy for non-adversarial inputs. Hence, there is a trade-off between robustness and accuracy for neural networks [38]. This issue can be addressed by combining a regular training objective with a robustness training objective [28, 16, 44].

1.2 Contributions

Our main contribution is a novel set-based training procedure that trains robust neural networks and drastically simplifies their subsequent formal robustness verification. Given a set of inputs with associated perturbance sets, we compute output sets and use the entire output sets for training. To our best knowledge, we present the first end-to-end set-based training approach for neural networks. Secondly, set-based training is made possible by an image enclosure using linear approximations. With our image enclosure, the output sets for an entire batch of inputs are efficiently computed using matrix operations on a GPU. Moreover, we show that the approximation errors by our image enclosure are always smaller or equal compared to an existing image enclosure. Thirdly, we evaluate our set-based training with neural networks of various sizes trained on three different datasets. We demonstrate the efficacy of set-based training and that set-based trained neural networks often outperform neural networks trained with state-of-the-art adversarial attacks.

1.3 Organisation

Sec. 2 introduces the required preliminaries. We provide an efficient image enclosure in Sec. 3, that we use for set-based training. In Sec. 4, we define our set-based training procedure, which we experimentally evaluate in Sec. 5. Finally, we conclude our findings in Sec. 6.

2 PRELIMINARIES

2.1 Notation

Lowercase letters denote vectors and uppercase letters denote matrices. The i-th entry of a vector x is denoted by $x_{(i)}$. For a matrix A, $A_{(i,j)}$ denotes the entry in the i-th row and the j-th column, $A_{(i,\cdot)}$ denotes the i-th row, and $A_{(\cdot,j)}$ the j-th column. The identity matrix is written as $I_n \in \mathbb{R}^{n\times n}$. We use $\mathbf{0}$ and $\mathbf{1}$ to represent the vector or matrix (with appropriate size) that contains only zeros or ones. The (horizontal) concatenation between two matrices $A \in \mathbb{R}^{m\times n_1}$ and $B \in \mathbb{R}^{m\times n_2}$ is denoted by $[\,A\;B\,]$. A square matrix with the entries of a given vector on its diagonal is returned by the operation $\text{diag} : \mathbb{R}^n \to \mathbb{R}^{n\times n}$. Given a matrix, the operation $\text{sign} : \mathbb{R}^{n\times m} \to \{-1, 0, 1\}^{n\times m}$ returns the sign of each entry of the matrix. We denote sets with uppercase calligraphic letters. For a set $\mathcal{S} \subset \mathbb{R}^n$, we denote its projection to the i-th dimension with $\mathcal{S}_{(i)}$. Given two sets $\mathcal{S}_1 \subset \mathbb{R}^n$ and $\mathcal{S}_2 \subset \mathbb{R}^m$, we denote the Cartesian product with $\mathcal{S}_1 \times \mathcal{S}_2 = \{\begin{bmatrix} s_1^\top & s_2^\top \end{bmatrix}^\top \mid s_1 \in \mathcal{S}_1,\, s_2 \in \mathcal{S}_2\}$, and if $n = m$, we write the Minkowski sum as $\mathcal{S}_1 \oplus \mathcal{S}_2 = \{s_1 + s_2 \mid s_1 \in \mathcal{S}_1,\, s_2 \in \mathcal{S}_2\}$. For $n \in \mathbb{N}$, $[n] = \{1, 2, \ldots, n\}$ denotes the set of all natural numbers up to n. We denote the natural logarithm with ln. An n-dimensional interval $\mathcal{I} \subset \mathbb{R}^n$ with bounds $l, u \in \mathbb{R}^n$ is denoted by $\mathcal{I} = [l, u]$. The continuous uniform distribution between $a \in \mathbb{R}$ and $b \in \mathbb{R}$ with $a \le b$ is written as $U(a, b)$. We denote a random variable X drawn from $U(a, b)$ by $X \sim U(a, b)$.

2.2 Feed-Forward Neural Networks

A feed-forward neural network consists of a sequence of $\kappa \in \mathbb{N}$ layers. A layer can either be a linear layer, which applies an affine map, or an activation layer, which applies a nonlinear activation function elementwise.

Definition 2.1 (Neural Network Layer, [4, Sec. 5.1]). The k-th layer of a neural network is defined as an operation $L_k : \mathbb{R}^{n_{k-1}} \to \mathbb{R}^{n_k}$,

$$h_k = L_k(h_{k-1}) = \begin{cases} W_k\, h_{k-1} + b_k & \text{if } k\text{-th layer is linear,} \\ \mu_k(h_{k-1}) & \text{otherwise,} \end{cases}$$

where n_{k-1} denotes the number of input neurons, n_k denotes the number of output neurons, $W_k \in \mathbb{R}^{n_k\times n_{k-1}}$ is a weight matrix, $b_k \in \mathbb{R}^{n_k}$ is a bias vector, and $\mu_k : \mathbb{R} \to \mathbb{R}$ is a nonlinear activation function.

The parameters θ of a neural network include all weight matrices and bias vectors from its linear layers. Forward propagation computes the output $y \in \mathbb{R}^{n_\kappa}$ of a neural network by propagating an input $x \in \mathbb{R}^{n_0}$ sequentially through all neural network layers.

Definition 2.2 (Forward Propagation, [4, Sec. 5.1]). The output $y \in \mathbb{R}^{n_\kappa}$ of a neural network for an input $x \in \mathbb{R}^{n_0}$ is computed by

$$\begin{aligned} h_0 &= x, \\ h_k &= L_k(h_{k-1}) \quad \text{for } k = 1, \ldots, \kappa, \\ y &= h_\kappa. \end{aligned}$$

The function $N_\theta(x) = y$ denotes the forward propagation through a neural network with parameters θ.

Training of Neural Networks. We only consider supervised training, where a neural network is trained with a training dataset $\mathcal{D} = \{(x_1, t_1), \ldots, (x_n, t_n)\}$, that contains inputs $x_i \in \mathbb{R}^{n_0}$ with associated target outputs $t_i \in \mathbb{R}^{n_\kappa}$. A loss function $E : \mathbb{R}^{n_\kappa} \times \mathbb{R}^{n_\kappa} \to \mathbb{R}$ measures how well a neural network predicts the target outputs. Typical loss functions are the half-squared error for regression tasks or the cross-entropy error for classification tasks.

Definition 2.3 (Half-Squared Error, [4, Sec. 5.2]). The half-squared error $E_{MSE} : \mathbb{R}^{n_\kappa} \times \mathbb{R}^{n_\kappa} \to \mathbb{R}$ is defined as

$$E_{MSE}(t, y) \coloneqq \frac{1}{2} \sum_{i=1}^{n_\kappa} (t_{(i)} - y_{(i)})^2.$$

Definition 2.4 (Cross-Entropy Error, [4, Sec. 5.2]). The cross-entropy error $E_{CE} : \mathbb{R}^{n_\kappa} \times \mathbb{R}^{n_\kappa} \to \mathbb{R}$ is defined as

$$E_{CE}(t, y) \coloneqq -\sum_{i=1}^{n_\kappa} t_{(i)}\, \ln(y_{(i)}) + (1 - t_{(i)})\, \ln(1 - y_{(i)}).$$

The training goal of a neural network is to find network parameters θ that minimize the total loss of the training dataset $\mathcal{D}$ [4, Sec. 5.2]:

$$\min_\theta \sum_{(x_i, t_i) \in \mathcal{D}} E(t_i, N_\theta(x_i)). \tag{1}$$

A popular algorithm to train a neural network is gradient descent [4, Sec. 5.2.4]: A random initial set of parameters $\theta^{(0)}$ is iteratively optimized using the gradient of the loss function [4, Eq. 5.41]. For the j-th iteration, let us introduce the gradient $g_k^{(j)}$ of the loss function E w.r.t. the output of the k-th layer h_k for the input $x \in \mathbb{R}^{n_0}$:

$$g_k^{(j)} \coloneqq \frac{\partial E(t, N_{\theta^{(j)}}(x))}{\partial h_k}. \tag{2}$$

In the j-th iteration, the weight matrix W_k and bias vector b_k of the k-th layer are updated as [4, Sec. 5.3]

$$\begin{aligned} W_k^{(j+1)} &\leftarrow W_k^{(j)} - \eta \frac{\partial E(t, N_{\theta^{(j)}}(x))}{\partial W_k^{(j)}} = W_k^{(j)} - \eta\, g_k^{(j)}\, h_{k-1}^\top, \\ b_k^{(j+1)} &\leftarrow b_k^{(j)} - \eta \frac{\partial E(t, N_{\theta^{(j)}}(x))}{\partial b_k^{(j)}} = b_k^{(j)} - \eta\, g_k^{(j)}, \end{aligned} \tag{3}$$

where $\eta \in \mathbb{R}_{>0}$ is the learning rate. For conciseness, we omit the iteration superscript from now on. The gradients g_k are efficiently computed with backpropagation [4, Sec. 5.3]: Utilizing the chain rule, the gradient g_κ of the last layer is propagated backward through all neural network layers.

Proposition 2.5 (Backpropagation, [4, Sec. 5.3]). *Let $y \in \mathbb{R}^{n_\kappa}$ be an output of a neural network with target $t \in \mathbb{R}^{n_\kappa}$. If the loss function E is the half-squared error or the cross-entropy error, then the gradients g_k are computed in reverse order as*

$$g_\kappa = y - t,$$
$$g_{k-1} = \begin{cases} W_k^\top\, g_k & \text{if } k\text{-th layer is linear,} \\ \operatorname{diag}\left(\frac{\mathrm{d}\mu_k(h_{k-1})}{\mathrm{d}h_{k-1}}\right) g_k & \text{otherwise,} \end{cases}$$

for all $k \in \{\kappa, \dots, 1\}$.

From now on, we refer to the (standard) neural network training as point-based training.

2.3 Set-Based Computation

We represent propagated sets by zonotopes. A zonotope is a convex set representation describing the Minkowski sum of a finite number of line segments.

Definition 2.6 (Zonotope, [13, Def. 1]). Given a center $c \in \mathbb{R}^n$ and a generator matrix $G \in \mathbb{R}^{n \times q}$, a zonotope $\mathcal{Z} \subset \mathbb{R}^n$ is defined as

$$\mathcal{Z} = \left\{ c + \sum_{j=1}^{q} \beta_{(j)}\, G_{(\cdot,j)} \;\middle|\; \beta \in [-1, 1]^q \right\} =: \langle c, G \rangle_Z.$$

Subsequently, we define several operations for zonotopes used in our training approach. We first present the tight enclosure of a zonotope by a multi-dimensional interval:

Proposition 2.7 (Interval Enclosure, [3, Prop. 2.2]). *A zonotope $\mathcal{Z} = \langle c, G \rangle_Z$ with $c \in \mathbb{R}^n$ and $G \in \mathbb{R}^{n \times q}$ is enclosed by the interval $[l, u] \supseteq \mathcal{Z}$, where*

$$l = c - \sum_{j=1}^{q} \left| G_{(\cdot,j)} \right|, \qquad u = c + \sum_{j=1}^{q} \left| G_{(\cdot,j)} \right|,$$

where $|\cdot|$ computes the elementwise absolute value. The time complexity of computing an interval enclosure is $O(n\,q)$.

Proposition 2.8 (Minkowski Sum, [3, Prop. 2.1 and Sec. 2.4]). *The Minkowski sum of a zonotope $\mathcal{Z} = \langle c, G \rangle_Z$ and an interval $\mathcal{I} = [l, u] \subset \mathbb{R}^n$ with $c, l, u \in \mathbb{R}^n$ and $G \in \mathbb{R}^{n \times q}$ is computed as*

$$\mathcal{Z} \oplus \mathcal{I} = \left\langle c + \frac{1}{2}(l + u), \left[G \quad \operatorname{diag}\left(\tfrac{1}{2}(u - l)\right) \right] \right\rangle_Z$$

and has time complexity $O(n^2)$.

Zonotopes are closed under linear maps.

Proposition 2.9 (Linear Map, [3, Sec. 2.4]). *The result of a linear map $f : \mathbb{R}^n \to \mathbb{R}^m$, $x \mapsto W x + b$ with $W \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ applied to a zonotope $\mathcal{Z} = \langle c, G \rangle_Z$ with $c \in \mathbb{R}^n$ and $G \in \mathbb{R}^{n \times q}$ is*

$$f(\mathcal{Z}) := \{ f(z) \mid z \in \mathcal{Z} \} = W\,\mathcal{Z} + b = \langle W c + b, W G \rangle_Z,$$

and has time complexity $O(m\,n\,q)$.

Determining the volume of a zonotope is computationally demanding [10]. However, a norm of a zonotope can effectively approximate the size of a zonotope [34, 7]. The interval norm computes the sum of the lengths of all edges of the interval enclosure of a zonotope.

Proposition 2.10 (Interval Norm, [2, Sec. 3.1]). *For a zonotope $\mathcal{Z} = \langle c, G \rangle_Z$, the interval norm is*

$$\|\mathcal{Z}\|_{\widetilde{I}} := \frac{1}{n}\, \mathbf{1}^\top \left| \begin{bmatrix} c & G \end{bmatrix} \right| \mathbf{1}.$$

Since the volume of a zonotope is invariant over translations, we exclude the center from the computation of the norm. The combined operation of translating the zonotope to the origin and computing the norm is denoted by

$$\|\mathcal{Z}\|_I := \|\langle \mathbf{0}, G \rangle_Z\|_{\widetilde{I}} = \frac{1}{n}\, \mathbf{1}^\top\, |G|\, \mathbf{1}. \tag{4}$$

2.4 Formal Verification of Neural Networks

In this work, we consider the robustness of neural networks for classification tasks: Each dimension of an output $y \in \mathbb{R}^{n_\kappa}$ corresponds to a classification label, and the dimension with the maximum value determines the predicted classification label for an input $x \in \mathbb{R}^{n_0}$. An input x is correctly classified by a neural network if the predicted classification label matches the classification label of the target output $t \in \mathbb{R}^{n_\kappa}$:

$$\arg\max_{k \in [\kappa]} y_{(k)} = \arg\max_{k \in [\kappa]} t_{(k)}. \tag{5}$$

We call a neural network (locally) robust for a given perturbance set if the neural network correctly classifies every input within the perturbance set. As a perturbance set we use the l_∞-ball of radius $\epsilon \in \mathbb{R}_{>0}$ around an input $x \in \mathbb{R}^{n_0}$:

Definition 2.11 (ϵ-Perturbance Set). For a perturbation radius $\epsilon \in \mathbb{R}_{>0}$ and an input $x \in \mathbb{R}^{n_0}$, the ϵ-perturbance set is defined as

$$\pi_\epsilon(x) := \left\langle x, \epsilon\, I_{n_0} \right\rangle_Z = \left\{ \tilde{x} \in \mathbb{R}^{n_0} \;\middle|\; \|\tilde{x} - x\|_\infty \le \epsilon \right\}.$$

We formally verify the robustness of a neural network by using set-based computations to compute its output set for an input set $\mathcal{X} = \pi_\epsilon(x)$. However, computing an exact output set of a feed-forward neural network is computationally hard; with only rectified linear unit (ReLU) activation functions, it has been shown to be NP-hard [19]. Thus, we compute an outer approximation $\widehat{\mathcal{Y}} \subset \mathbb{R}^{n_\kappa}$ of the true output set $\mathcal{Y}^* := \{ N_\theta(x) \mid x \in \mathcal{X} \} \subseteq \widehat{\mathcal{Y}}$ of a neural network N_θ for an input set $\mathcal{X} \subset \mathbb{R}^{n_0}$. If $\widehat{\mathcal{Y}}$ does not intersect with a region of unsafe outputs $\mathcal{U}$, we have formally verified that the neural network only computes safe outputs for every input $x \in \mathcal{X}$. For a classification task with target $t \in \mathbb{R}^{n_\kappa}$, the unsafe set contains every incorrect classification [23, Prop. B.2]:

$$\mathcal{U} := \left\{ y \in \mathbb{R}^{n_\kappa} \;\middle|\; (I_{n_\kappa} - \mathbf{1}\, e_l)\, y \not\le \mathbf{0} \right\}, \tag{6}$$

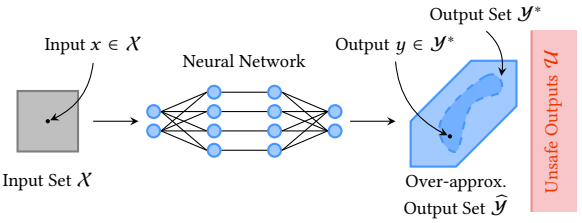

Figure 1: Verifying the local robustness of a neural network.

where $l := \arg\max_{k \in [\kappa]} t_{(k)}$ and e_l is the l-th standard basis vector. Fig. 1 illustrates the robustness verification of a neural network. To compute an outer approximation $\widehat{\mathcal{Y}}$, we evaluate the operations L_k (Def. 2.1) by sets. We compute the output set of a linear layer for an input set with a linear map [23, Sec. 2.4]. Zonotopes are not closed under nonlinear maps; hence, we enclose the output set of an activation layer by an image enclosure summarized in Alg. 1 [23, Prop. 2.14]. We first compute an upper bound and a lower bound for each dimension of the input set (Line 3). The activation function is approximated within the computed bounds using a polynomial regression (Line 4). To ensure the soundness of the approximations, a bound on the approximation errors is computed and added to the result (Lines 5–7). Using Alg. 1, we define a set-based forward propagation.

Algorithm 1: Image enclosure of an activation layer [23].

1 **function** `enclose`$(L_k, \mathcal{H}_{k-1})$
2 **for** $i \leftarrow 1$ **to** n_k **do**
3 Find bounds $\mathcal{I}_{k-1(i)}$ of $\mathcal{H}_{k-1(i)}$ // Prop. 2.7
4 Find polynomial approximation $p_{k,i}$ of μ_k // [31]
5 $d_{k(i)} \leftarrow \max_{x \in \mathcal{I}_{k-1(i)}} \left|\mu_k(x) - p_{k,i}(x)\right| + \delta$ // [23, Eq. 18]
6 $\widetilde{\mathcal{H}}_{k(i)} \leftarrow$ evaluate $p_{k,i}$ with $\mathcal{H}_{k-1(i)}$ // [23, Sec. 3]
7 $\mathcal{H}_{k(i)} \leftarrow \widetilde{\mathcal{H}}_{k(i)} \oplus [-d_{k(i)}, d_{k(i)}]$ // Prop. 2.8
8 $\mathcal{H}_k \leftarrow \mathcal{H}_{k(1)} \times \cdots \times \mathcal{H}_{k(n_k)}$
9 **return** $\mathcal{H}_k$

Proposition 2.12 (Set-Based Forward Prop., [23, Sec. 2.4]). *For an input set $\mathcal{X} \subset \mathbb{R}^{n_0}$, an outer approximation of the output set $\widehat{\mathcal{Y}} \subset \mathbb{R}^{n_\kappa}$ of a neural network can be computed as*

$$\mathcal{H}_0 = \mathcal{X},$$
$$\mathcal{H}_k = L_k(\mathcal{H}_{k-1}),$$
$$\widehat{\mathcal{Y}} = \mathcal{H}_\kappa \supseteq \mathcal{Y}^* := \{N_\theta(x) \mid x \in \mathcal{X}\},$$

where for $k = 1, \ldots, \kappa$,

$$L_k(\mathcal{H}_{k-1}) = \begin{cases} W_k\, \mathcal{H}_{k-1} + b_k & \text{if } k\text{-th layer is linear,} \\ \texttt{enclose}(L_k, \mathcal{H}_{k-1}) & \text{otherwise,} \end{cases}$$

and the operation `enclose` *is specified in Alg. 1.*

2.5 Problem Statement

The training goal for a robust neural network is to minimize the worst-case loss within the ϵ-perturbance set of each input in the training dataset [27, Sec. 2]:

$$\min_\theta \sum_{(x_i,t_i)\in\mathcal{D}} \max_{\tilde{x}_i \in \pi_\epsilon(x_i)} E(t_i, N_\theta(\tilde{x}_i)). \tag{7}$$

In this work, we derive a end-to-end set-based training procedure that uses set-based computations (i) to train robust neural networks and (ii) to simplify the subsequent robustness verification of the trained neural networks.

3 FAST IMAGE ENCLOSURE OF ACTIVATION FUNCTIONS

The training of neural networks requires many forward propagations, which are costly for set-based forward propagations due to the image enclosures of activation layers. This makes sampling-based methods [23, 21] for the image enclosure impractical. In contrast, [35] derives fast analytical solutions for the approximation errors of a specific linear approximation of s-shaped activation functions. However, these linear approximations create large approximation errors. To address this issue, we derive analytical solutions for the approximation errors of an arbitrary monotonically increasing linear approximation for three typical activation functions: ReLU (Sec. 3.1), hyperbolic tangent (Sec. 3.2), and logistic sigmoid (Sec. 3.2). Secondly, we provide a linear approximation with approximation errors that are smaller or equal to [35, Thm. 3.2] while being equally fast to compute. For the remainder of this section, let $\mu : \mathbb{R} \to \mathbb{R}$ be a monotonically increasing function which is approximated within an interval $\mathcal{I} = [l, u] \subset \mathbb{R}$.

Definition 3.1 (Linear Approximation). Within the interval $\mathcal{I}$, we approximate μ by a linear function $p(x) := m\,x + a$, where

$$m := \frac{\mu(u) - \mu(l)}{u - l}, \qquad a := \mu\left(\frac{u+l}{2}\right) - m\,\frac{u+l}{2}.$$

The approximation errors of p are given by the largest lower distance and upper distance between μ and p (see Fig. 2).

Definition 3.2 (Approximation Errors). The approximation errors of p (Def. 3.1) for μ within the interval $\mathcal{I}$ are defined as

$$\underline{d} := \min_{x \in \mathcal{I}} \mu(x) - p(x), \qquad \overline{d} := \max_{x \in \mathcal{I}} \mu(x) - p(x).$$

3.1 Approximation Errors for ReLU

The ReLU is defined as $\mathrm{ReLU}(x) := \max(0, x)$. Since ReLU is piecewise linear, it suffices to evaluate $\mathrm{ReLU}(x) - p(x)$ at $x = 0$ and the bounds $x \in \{l, u\}$.

Proposition 3.3 (Approximation Errors for ReLU). *The approximation errors of p for ReLU are computed as*

$$\underline{d} = \min_{x \in \mathcal{P}} \mathrm{ReLU}(x) - p(x), \qquad \overline{d} = \max_{x \in \mathcal{P}} \mathrm{ReLU}(x) - p(x),$$

where $\mathcal{P} = \{l, 0, u\} \cap \mathcal{I}$.

Proof. ReLU is monotonic for $[l, 0]$ and $[0, u]$. Thus, the approximation errors are found at $x = 0$ or the bounds $x \in \{l, u\}$. □

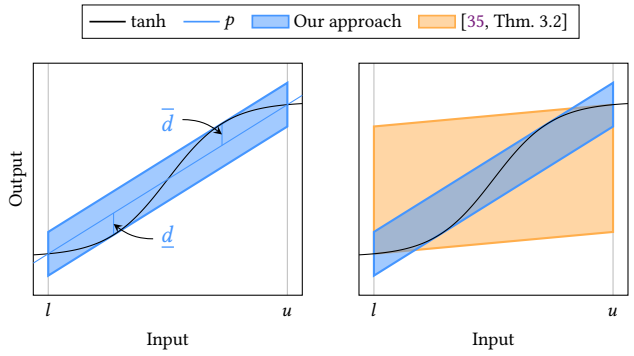

Figure 2: Image enclosure of hyperbolic tangent: (left) Our linear approximation and approximation errors; (right) Comparison of our image enclosure and the image enclosure by [35, Thm. 3.2].

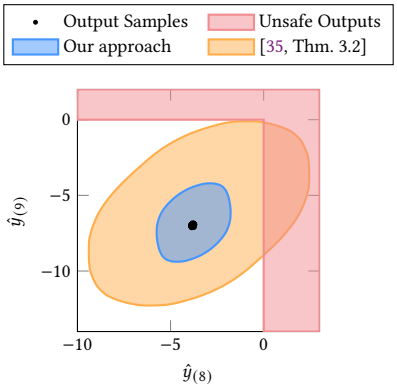

Figure 3: Comparison of the output set of a neural network (nn-small trained point-based on F-MNIST with the input set $\pi_\epsilon(x)$ where $\epsilon = 0.005$) computed with different image enclosures: The output set computed with our image enclosure is significantly smaller and does not intersect the unsafe region, while the output set computed with the image enclosure by [35, Thm. 3.2] does intersect the unsafe region.

3.2 Approximation Errors for Hyperbolic Tangent and Logistic Sigmoid

We can efficiently compute the approximation errors for differentiable activation functions by returning the minimum and maximum values of the finite set of extreme points of $\mu(x) - p(x)$. If the extreme points are not contained within the interval, we include the boundaries of the interval. For the hyperbolic tangent and logistic sigmoid, we provide the set $\mathcal{P}$ of all extreme points of $\mu(x) - p(x)$.

Proposition 3.4 (Approximation Errors for Hyperbolic Tangent). *The approximation errors of p for* tanh *are*

$$\underline{d} = \min_{x \in \mathcal{P}} \tanh(x) - p(x), \qquad \overline{d} = \max_{x \in \mathcal{P}} \tanh(x) - p(x),$$

where $\mathcal{P} = \left\{ \pm \tanh^{-1}\left(\sqrt{1-m}\right), l, u \right\} \cap \mathcal{I}$.

Proof. The derivative of the hyperbolic tangent is $1 - \tanh(x)^2$. We demand that the derivative of $\tanh(x) - p(x)$ is 0 and simplify the terms:

$$\begin{aligned} & 0 = \frac{\mathrm{d}}{\mathrm{d}x}(\tanh(x) - p(x)) \\ \Leftrightarrow \quad & 0 = 1 - \tanh(x)^2 - m \\ \Leftrightarrow \quad & \tanh(x) = \pm\sqrt{1-m} \\ \Leftrightarrow \quad & x = \pm \tanh^{-1}\left(\sqrt{1-m}\right). \end{aligned}$$

□

The logistic sigmoid is defined as $\sigma(x) := \frac{1}{1+e^{-x}}$ [14, Sec. 6.3.2].

Proposition 3.5 (Approximation Errors for Logistic Sigmoid). *The approximation errors of p for σ are*

$$\underline{d} = \min_{x \in \mathcal{P}} \sigma(x) - p(x), \qquad \overline{d} = \max_{x \in \mathcal{P}} \sigma(x) - p(x),$$

where $\mathcal{P} = \left\{ \pm 2 \tanh^{-1}\left(\sqrt{1-4m}\right), l, u \right\} \cap \mathcal{I}$.

Proof. It holds that $\sigma(x) = 0.5\,(\tanh(0.5\,x) + 1)$ [14, Sec. 6.3.2]. We demand that derivative of $\sigma(x) - p(x)$ is 0 and simplify the terms:

$$\begin{aligned} & 0 = \frac{\mathrm{d}}{\mathrm{d}x}(\sigma(x) - p(x)) \\ \Leftrightarrow \quad & 0 = \frac{\mathrm{d}}{\mathrm{d}x}(0.5\,(\tanh(0.5\,x) + 1)) - m \\ \Leftrightarrow \quad & \tanh(0.5\,x) = \pm\sqrt{1-4\,m} \\ \Leftrightarrow \quad & x = \pm 2\,\tanh^{-1}\left(\sqrt{1-4\,m}\right). \end{aligned}$$

□

While the approach in [35, Thm. 3.2] only works for a specific linear approximation, our approach works for any (monotonically increasing) linear approximation. In addition, we prove that the approximation errors of our linear approximation (Def. 3.1) are always smaller or equal to the approximation errors by [35] w.r.t. the area in the input-output plane (see Fig. 2) measuring the integrated approximation error over $\mathcal{I}$:

$$A([\underline{d}, \overline{d}], \mathcal{I}) := (l - u)\,(\overline{d} - \underline{d}). \tag{8}$$

Theorem 3.6. *Let μ be an s-shaped function, and let $\mathcal{I} = [l, u]$ be an interval. Moreover, let $\underline{d}$ and $\overline{d}$ be the approximation errors of p as defined in Def. 3.1 and 3.2, and let d_S be the approximation error by [35, Thm. 3.2]. It holds that*

$$A([\underline{d}, \overline{d}], \mathcal{I}) \leq A([-d_S, d_S], \mathcal{I}).$$

Proof. See appendix. □

Fig. 3 shows an instance where the smaller approximation errors by our image enclosure enable the verification of a neural network that is not possible with the image enclosure by [35, Thm. 3.2].

3.3 Implementation

Alg. 2 implements our image enclosure. First, we compute the interval bounds of the input set (Line 2). For each neuron, the linear approximation (Line 4) and the approximation errors (Line 5) are

computed. The linear approximations are applied to the input set (Line 6), and finally the approximation errors (Line 7) are added. The time complexity of Alg. 2 is polynomial. We use the time complexity of Alg. 2 to later derive the time complexity of our training approach.

Proposition 3.7 (Time Complexity of Alg. 2). *For an input set $\mathcal{H}_{k-1} = \langle c, G \rangle_Z$ with $c \in \mathbb{R}^n$ and $G \in \mathbb{R}^{n \times q}$, Alg. 2 has time complexity $O(n^2\, q)$ w.r.t. the number of input dimensions n and the number of generators q.*

Proof. Finding the interval bounds of $\mathcal{H}_{k-1}$ (Line 2) takes time $O(n\, q)$ (Prop. 2.7). Computing the linear approximation (Line 4) and the approximation errors (Line 5) for each neuron takes constant time; hence, the loop takes time $O(n)$. The linear map of $\mathcal{H}_{k-1}$ (Line 6) takes time $O(n^2\, q)$ (Prop. 2.9). Adding the approximation errors (Line 7) takes time $O(n^2)$. Thus, in total we have $O(n\, q) + O(n) + O(n^2\, q) + O(n^2) = O(n^2\, q)$. □

Algorithm 2: Fast image enclosure of an activation layer.

```
1 function fastEnclose(L_k, H_{k-1})
2     Find bounds I_{k-1} of H_{k-1}                    // Prop. 2.7
3     for i ← 1 to n_k do
4         Compute linear approx. m_{k(i)} x + a_{k(i)}  // Def. 3.1
5         Compute approx. errors d_{k(i)}, d̄_{k(i)}     // Prop. 3.3 to 3.5
6     H̃_k ← diag(m_k) H_{k-1} + a_k                     // Prop. 2.9
7     H_k ← H̃_k ⊕ [d_k, d̄_k]                            // Prop. 2.8
8     return H_k, m_k
```

The image enclosure of an s-shaped function like hyperbolic tangent or logistic sigmoid by [21, Sec. 3.2] fits a polynomial to the activation function using polynomial regression and uses many evenly distributed samples along the activation function to bound the approximation errors (see Alg. 1). Alg. 2 no longer uses a polynomial regression or requires sampling to compute the approximation errors. Moreover, each loop iteration of Alg. 2 is independent, and the entire loop can be efficiently computed in a batch-wise fashion using matrix operations on a GPU. The results of this section obviously also benefit the set-based verification of neural networks.

4 SET-BASED TRAINING OF NEURAL NETWORKS

We present a novel set-based training procedure for neural networks. Intuitively, we replace each point-based training step with a set-based training step and make adjustments where necessary. In each training iteration, we (i) compute a set of losses containing a loss for every input of an ϵ-perturbance set (Def. 2.11), (ii) derive a set-based backpropagation that computes sets of gradients, and (iii) aggregate the sets of gradients to update the values for the parameters of the neural network.

4.1 Set-Based Loss

First, we define a set-based loss function returning a set of losses for an entire set of outputs. In our work, we define a set-based loss function $\widetilde{E} : \mathbb{R}^{n_\kappa} \times 2^{\mathbb{R}^{n_\kappa}} \to 2^{\mathbb{R}^{n_\kappa}}$ so that it combines (a) the set-based evaluation of a point-based loss with (b) the interval norm of the output set (4). The set-based loss function uses a hyperparameter $\tau \in [0, 1]$ to weigh the set-based evaluation of the point-based loss and the interval norm. The included interval norm minimizes the size of the output sets, thereby increasing the robustness of the trained neural network.

Definition 4.1 (Set-Based Loss). Given a (point-based) loss function $E : \mathbb{R}^{n_\kappa} \times \mathbb{R}^{n_\kappa} \to \mathbb{R}$, we define a set-based loss function as

$$\widetilde{E}(t, \widehat{\mathcal{Y}}) := (1 - \tau)\, E(t, \widehat{\mathcal{Y}}) + \frac{\tau}{\epsilon} \left\| \widehat{\mathcal{Y}} \right\|_I,$$

where $E(t, \widehat{\mathcal{Y}}) = \left\{ E(t, y) \;\middle|\; y \in \widehat{\mathcal{Y}} \right\}$ is the set-based evaluation of E, $\epsilon \in \mathbb{R}_{>0}$ is the input perturbation radius, and $\left\| \widehat{\mathcal{Y}} \right\|_I$ is the interval norm of $\widehat{\mathcal{Y}}$ according to (4).

To make tuning the hyperparameter τ easier, the interval norm in Def. 4.1 is normalized with the input perturbation radius $\epsilon \in \mathbb{R}_{>0}$. The normalization is derived from the ratio of the interval norms of the output set and the input set, which is an ϵ-perturbance set:

$$\frac{\left\| \widehat{\mathcal{Y}} \right\|_I}{\| \pi_\epsilon(x) \|_I} \overset{\text{Def. 2.11}}{=} \frac{\left\| \widehat{\mathcal{Y}} \right\|_I}{\epsilon}. \tag{9}$$

Similar to point-based training, we use the gradient of the set-based loss function to update the parameters of a neural network. The gradient of a set-based loss is a set that contains a gradient for every input of the input set. Computing the set-based evaluation of the point-based loss might be hard or impossible, e.g., computing a set-based cross-entropy error is hard due to the logarithm (see Def. 2.4). However, we only need the gradient of the set-based loss for training. If the point-based loss is the half-squared error or the cross-entropy error (Def. 2.3 and 2.4), the gradient is just the difference between the output and the target (Prop. 2.5):

$$\begin{aligned} \frac{\partial E(t, \widehat{\mathcal{Y}})}{\partial \widehat{\mathcal{Y}}} &= \left\{ \frac{\partial E(t, y)}{\partial y} \;\middle|\; y \in \widehat{\mathcal{Y}} \right\} = \left\{ y - t \;\middle|\; y \in \widehat{\mathcal{Y}} \right\} \\ &= \left\langle c_{\widehat{\mathcal{Y}}} - t, G_{\widehat{\mathcal{Y}}} \right\rangle_Z, \end{aligned} \tag{10}$$

where $\widehat{\mathcal{Y}} = \left\langle c_{\widehat{\mathcal{Y}}}, G_{\widehat{\mathcal{Y}}} \right\rangle_Z$.

The second term of the set-based loss function is the interval norm of the output set. The gradient of the interval norm is a zonotope, where the center is the derivative w.r.t. the center and the generator matrix is the derivative w.r.t. the generator matrix.

Proposition 4.2 (Gradient of Interval Norm). *The gradient of the interval norm is*

$$\frac{\partial \| \mathcal{Z} \|_I}{\partial \mathcal{Z}} := \left\langle \frac{\partial \| \mathcal{Z} \|_I}{\partial c}, \frac{\partial \| \mathcal{Z} \|_I}{\partial G} \right\rangle_Z = \frac{1}{n} \langle \mathbf{0}, \operatorname{sign}(G) \rangle_Z,$$

where $\mathcal{Z} = \langle c, G \rangle_Z \subset \mathbb{R}^n$.

Proof. The gradient of the absolute value $|\cdot|$ is the sign function. Thus,

$$\frac{\partial \| \mathcal{Z} \|_I}{\partial \mathcal{Z}} = \left\langle \frac{\partial \| \mathcal{Z} \|_I}{\partial c}, \frac{\partial \| \mathcal{Z} \|_I}{\partial G} \right\rangle_Z = \frac{1}{n} \langle \mathbf{0}, \operatorname{sign}(G) \rangle_Z.$$ □

By combining (10) and Prop. 4.2, we can compute the gradient of a set-based loss function:

Proposition 4.3 (Gradient of Set-Based Loss). *If the point-based loss E is the half-squared error or the cross-entropy error (Def. 2.3 and 2.4), then the gradient of the set-based loss function $\widetilde{E}$ is*

$$\frac{\partial \widetilde{E}(t, \widehat{\mathcal{Y}})}{\partial \widehat{\mathcal{Y}}} = \left\langle (1-\tau)\,(c_{\widehat{\mathcal{Y}}} - t), (1-\tau)\,G_{\widehat{\mathcal{Y}}} + \frac{\tau}{\epsilon\, n_\kappa}\ \mathrm{sign}(G_{\widehat{\mathcal{Y}}}) \right\rangle_Z,$$

where $\widehat{\mathcal{Y}} = \left\langle c_{\widehat{\mathcal{Y}}}, G_{\widehat{\mathcal{Y}}} \right\rangle_Z$.

Proof. This directly follows from (10) and Prop. 4.2. □

To have a end-to-end set-based training process, we extend the backpropagation (Prop. 2.5) through neural networks to sets.

4.2 Set-Based Backpropagation

In this section, we lift the point-based backpropagation to a set-based evaluation. For every layer of the neural network, the set-based backpropagation computes the gradient of the set-based loss function $\widetilde{E}$ w.r.t. the output set $\mathcal{H}_k$:

$$\mathcal{G}_k := \frac{\partial \widetilde{E}(t, \widehat{\mathcal{Y}})}{\partial \mathcal{H}_k} = \left\{ \frac{\partial \widetilde{E}(t, \widehat{\mathcal{Y}})}{\partial h_k} \;\middle|\; h_k \in \mathcal{H}_k \right\}. \tag{11}$$

The set-based backpropagation of linear layers is straightforward as it just applies a linear map (Prop. 2.5). However, the backpropagation of activation layers requires the gradient of its nonlinear activation function (Prop. 2.5). Since zonotopes are not closed under nonlinear maps, we approximate the gradient of the activation function μ_k with the slope $m_{k(i)}$ of its linear approximation $p_{k,i}$ (see Alg. 2) obtained during the set-based forward propagation. For the i-th neuron, we have

$$m_{k(i)} = \frac{\partial p_{k,i}(\mathcal{H}_{k-1(i)})}{\partial \mathcal{H}_{k-1(i)}} \approx \frac{\partial \mu_k(\mathcal{H}_{k-1(i)})}{\partial \mathcal{H}_{k-1(i)}}. \tag{12}$$

Using this approximation, we can formulate the set-based backpropagation:

Proposition 4.4 (Set-Based Backpropagation). *Let $\widehat{\mathcal{Y}} \subset \mathbb{R}^{n_\kappa}$ be an output set of a neural network with target $t \in \mathbb{R}^{n_\kappa}$. The gradients $\mathcal{G}_k$ are computed in reverse order as*

$$\mathcal{G}_\kappa = \frac{\partial \widetilde{E}(t, \widehat{\mathcal{Y}})}{\partial \widehat{\mathcal{Y}}},$$

$$\mathcal{G}_{k-1} = \begin{cases} W_k^\top\, \mathcal{G}_k & \text{if } k\text{-th layer is linear,} \\ \mathrm{diag}(m_k)\, \mathcal{G}_k & \text{otherwise,} \end{cases}$$

for all $k \in \{\kappa, \dots, 1\}$. If the k-th layer is an activation layer, then the vector $m_k \in \mathbb{R}^{n_k}$ denotes the slopes of the linear approximations of that layer (Alg. 2).

Proof. See appendix. □

Fig. 4 illustrates the computations during a set-based forward propagation and set-based backpropagation.

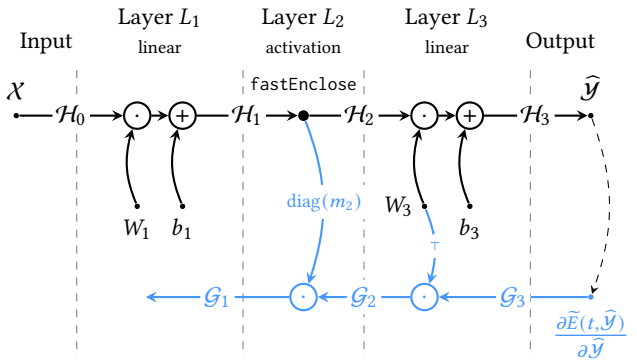

Figure 4: Illustration of a set-based forward propagation (black) and set-based backpropagation (blue).

4.3 Set-Based Update of Weights and Biases

We describe now how we use the set of gradients $\mathcal{G}_k$ and the set of inputs $\mathcal{H}_{k-1}$ to update the weights and biases of a linear layer. Our main idea is to compute the outer product in (3) in a set-based fashion using $\mathcal{G}_k\, \mathcal{H}_{k-1}^\top \subset \mathbb{R}^{n_k \times n_{k-1}}$. However, there are dependencies between the zonotopes, as the set of gradients $\mathcal{G}_k$ are ultimately computed from the set of inputs $\mathcal{H}_{k-1}$. We explicitly keep track of these dependencies using symbolic zonotopes [8], which assign each generator a label. Let $\mathcal{G}_k = \langle c_\mathcal{G}, G_\mathcal{G} \rangle_Z$ and $\mathcal{H}_{k-1} = \langle c_\mathcal{H}, G_\mathcal{H} \rangle_Z$ with $c_\mathcal{G} \in \mathbb{R}^{n_k}$, $G_\mathcal{G} \in \mathbb{R}^{n_k \times q}$, $c_\mathcal{H} \in \mathbb{R}^{n_{k-1}}$, and $G_\mathcal{H} \in \mathbb{R}^{n_{k-1} \times r}$. We assign each column of the generator matrices $G_\mathcal{G}$ and $G_\mathcal{H}$ a label, which we denote by $G_\mathcal{G}^{|l_\mathcal{G}}$ and $G_\mathcal{H}^{|l_\mathcal{H}}$, where $l_\mathcal{G} \in \mathbb{N}^q$ and $l_\mathcal{H} \in \mathbb{N}^r$ are labels. We use the labels to compute a Cartesian product that respects the dependencies between the sets $\mathcal{G}_k$ and $\mathcal{H}_{k-1}$ [25, Sec. II.C]:

$$\mathcal{G}_k \times \mathcal{H}_{k-1} = \left\langle \begin{bmatrix} c_\mathcal{G} \\ c_\mathcal{H} \end{bmatrix}, \begin{bmatrix} \mathbf{0} & G_\mathcal{G}^{|l_\mathcal{G} \cap l_\mathcal{H}} & G_\mathcal{G}^{|l_\mathcal{G} \setminus l_\mathcal{H}} \\ G_\mathcal{H}^{|l_\mathcal{H} \setminus l_\mathcal{G}} & G_\mathcal{H}^{|l_\mathcal{G} \cap l_\mathcal{H}} & \mathbf{0} \end{bmatrix} \right\rangle_Z. \tag{13}$$

The generators with common labels are concatenated vertically, whereas those with unique labels are padded with zeros. Using the Cartesian product, we define a set-based outer product that respects the dependencies between $\mathcal{G}_k$ and $\mathcal{H}_{k-1}$:

$$\mathcal{G}_k\, \mathcal{H}_{k-1}^\top := \left\{ g_k\, h_{k-1}^\top \;\middle|\; \begin{bmatrix} g_k^\top & h_{k-1}^\top \end{bmatrix}^\top \in \mathcal{G}_k \times \mathcal{H}_{k-1} \right\}. \tag{14}$$

To update the weight matrix and the bias vector, we could simply sample the set-based outer product. However, by sampling, we lose much information contained in the gradient set and the input set. While we could increase the effectiveness by taking more samples, we want to use all gradients and inputs to update the weights and biases. Hence, we aggregate the entire set-based outer product by computing its expected value, where we see the factors of the generators β as random variables. First, we simplify the computation of the set-based outer product by exploiting the dependencies between the gradients $\mathcal{G}_k$ and the inputs $\mathcal{H}_{k-1}$:

Proposition 4.5. *The set-based outer product between $\mathcal{G}_k$ and $\mathcal{H}_{k-1}$ is computed as*

$$\mathcal{G}_k\, \mathcal{H}_{k-1}^\top = \left\{ \mathbf{w}_k(\beta) \;\middle|\; \beta \in [-1, 1]^q \right\},$$

where

$$\mathbf{b}_k(\beta) \coloneqq \begin{cases} c_{\mathcal{G}} + G_{\mathcal{G}}^{|l_{\mathcal{G}}} \beta & \text{if } \beta \in [-1,1]^q, \\ \mathbf{0} & \text{otherwise,} \end{cases}$$

$$\mathbf{w}_k(\beta) \coloneqq \begin{cases} \mathbf{b}_k(\beta) \left(c_{\mathcal{H}} + \begin{bmatrix} G_{\mathcal{H}}^{|l_{\mathcal{H}}} & \mathbf{0} \end{bmatrix} \beta \right)^\top & \text{if } \beta \in [-1,1]^q, \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

Proof. We make two observations: (a) The image enclosure of an activation layer adds new generators for the approximation errors (Alg. 2); (b) the ordering of the generators is not changed during a set-based forward propagation or a set-based backpropagation. Hence, $l_{\mathcal{H}} \setminus l_{\mathcal{G}}$ is empty and $l_{\mathcal{G}} \cap l_{\mathcal{H}} = l_{\mathcal{H}}$:

$$\begin{bmatrix} \mathbf{0} & G_{\mathcal{G}}^{|l_{\mathcal{G}} \cap l_{\mathcal{H}}} & G_{\mathcal{G}}^{|l_{\mathcal{G}} \setminus l_{\mathcal{H}}} \\ G_{\mathcal{H}}^{|l_{\mathcal{H}} \setminus l_{\mathcal{G}}} & G_{\mathcal{H}}^{|l_{\mathcal{G}} \cap l_{\mathcal{H}}} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} G_{\mathcal{G}}^{|l_{\mathcal{H}}} & G_{\mathcal{G}}^{|l_{\mathcal{G}} \setminus l_{\mathcal{H}}} \\ G_{\mathcal{H}}^{|l_{\mathcal{H}}} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} G_{\mathcal{G}}^{|l_{\mathcal{G}}} \\ G_{\mathcal{H}}^{|l_{\mathcal{H}}} & \mathbf{0} \end{bmatrix}.$$

Thus, for $\begin{bmatrix} g_k^\top & h_{k-1}^\top \end{bmatrix}^\top \in \mathcal{G}_k \times \mathcal{H}_{k-1}$, with some $\beta \in [-1,1]^q$:

$$g_k \, h_{k-1}^\top \overset{(13)}{=} \left(c_{\mathcal{G}} + G_{\mathcal{G}}^{|l_{\mathcal{G}}} \beta \right) \left(c_{\mathcal{H}} + \begin{bmatrix} G_{\mathcal{H}}^{|l_{\mathcal{H}}} & \mathbf{0} \end{bmatrix} \beta \right)^\top = \mathbf{w}_k(\beta). \qquad \square$$

For a specific $\beta \in [-1,1]^q$, $\mathbf{w}_k(\beta)$ and $\mathbf{b}_k(\beta)$ are samples:

$$\mathbf{w}_k(\beta) \in \mathcal{G}_k \, \mathcal{H}_{k-1}^\top, \qquad \mathbf{b}_k(\beta) \in \mathcal{G}_k. \tag{15}$$

We aggregate all elements of the set-based outer product by computing the expected value of $\mathbf{w}_k$ for an assumed probability distribution of the factors β; to update the bias vectors, we compute the expected value of $\mathbf{b}_k$:

$$W_k \leftarrow W_k - \eta \, \mathrm{E}[\mathbf{w}_k(\beta)], \qquad b_k \leftarrow b_k - \eta \, \mathrm{E}[\mathbf{b}_k(\beta)]. \tag{16}$$

Prop. 4.6 computes the expected values of $\mathbf{w}_k$ and $\mathbf{b}_k$ based on a probability distribution of the factors β. To simplify matters, we assume that the expected value of a factor is 0, which is reasonable as zonotopes are point-symmetric.

Proposition 4.6. *Let β be a random variable with*

$$\mathrm{E}[\beta] = \mathbf{0}, \tag{17}$$

the expected values of $\mathbf{w}_k$ and $\mathbf{b}_k$ are

$$\mathrm{E}[\mathbf{w}_k(\beta)] = c_{\mathcal{G}} \, c_{\mathcal{H}}^\top + G_{\mathcal{G}}^{|l_{\mathcal{G}}} \, \mathrm{E}\left[\beta \, \beta^\top\right] \begin{bmatrix} G_{\mathcal{H}}^{|l_{\mathcal{H}}} & \mathbf{0} \end{bmatrix}^\top,$$
$$\mathrm{E}[\mathbf{b}_k(\beta)] = c_{\mathcal{G}}.$$

Proof. Using (17) we have

$$\mathrm{E}\left[\mathbf{b}_k(\beta)_{(j)}\right] = c_{\mathcal{G}(j)} + G_{\mathcal{G}(j,\cdot)}^{|l_{\mathcal{G}}} \, \mathrm{E}[\beta] = c_{\mathcal{G}(j)}.$$

Moreover, we show,

$$\begin{aligned} \mathrm{E}\left[\mathbf{w}_k(\beta)_{(i,j)}\right] &\overset{\text{Prop. 4.5}}{=} \mathrm{E}\left[\mathbf{b}_k(\beta)_{(i)} \left(c_{\mathcal{H}(j)} + \begin{bmatrix} G_{\mathcal{H}(j,\cdot)}^{|l_{\mathcal{H}}} & \mathbf{0} \end{bmatrix} \beta \right)\right] \\ &= c_{\mathcal{G}(i)} \, c_{\mathcal{H}(j)} + \left(c_{\mathcal{G}(i)} \begin{bmatrix} G_{\mathcal{H}(j,\cdot)}^{|l_{\mathcal{H}}} & \mathbf{0} \end{bmatrix} + c_{\mathcal{H}(j)} \, G_{\mathcal{G}(i,\cdot)}^{|l_{\mathcal{G}}} \right) \mathrm{E}[\beta] \\ &\quad + G_{\mathcal{G}(i,\cdot)}^{|l_{\mathcal{G}}} \, \mathrm{E}\left[\beta \, \beta^\top\right] \begin{bmatrix} G_{\mathcal{H}(j,\cdot)}^{|l_{\mathcal{H}}} & \mathbf{0} \end{bmatrix}^\top \\ &\overset{(17)}{=} c_{\mathcal{G}(i)} \, c_{\mathcal{H}(j)} + G_{\mathcal{G}(i,\cdot)}^{|l_{\mathcal{G}}} \, \mathrm{E}\left[\beta \, \beta^\top\right] \begin{bmatrix} G_{\mathcal{H}(j,\cdot)}^{|l_{\mathcal{H}}} & \mathbf{0} \end{bmatrix}^\top. \end{aligned} \qquad \square$$

For now, we assume an independent and uniform distribution of the factors $\beta \sim U(-1,1)^q$.

Corollary 4.7. *Assuming $\beta \sim U(-1,1)^q$, the expected values of $\mathbf{w}_k$ and $\mathbf{b}_k$ are*

$$\mathrm{E}[\mathbf{w}_k(\beta)] = c_{\mathcal{G}} \, c_{\mathcal{H}}^\top + \frac{1}{3} G_{\mathcal{G}}^{|l_{\mathcal{G}}} \begin{bmatrix} G_{\mathcal{H}}^{|l_{\mathcal{H}}} & \mathbf{0} \end{bmatrix}^\top, \qquad \mathrm{E}[\mathbf{b}_k(\beta)] = c_{\mathcal{G}}.$$

Proof. We use Prop. 4.6: The expected value of the uniform distribution $U(-1,1)$ is 0, hence $\mathrm{E}[\beta] = \mathbf{0}$ [9, Sec. 5.2]. If $i \neq j$, then β_i and β_j are independent and hence $\mathrm{E}\left[\beta_i \, \beta_j\right] = \mathrm{E}[\beta_i] \, \mathrm{E}\left[\beta_j\right] = 0$. Moreover, if $i = j$, then $\mathrm{E}\left[\beta_i \, \beta_j\right] = \mathrm{E}\left[\beta_i^2\right] = \frac{1}{3}$ [9, Sec. 7.3]. Thus, ultimately, we have $\mathrm{E}\left[\beta \, \beta^\top\right] = \frac{1}{3} I_q$. $\square$

The uniform distribution of the factors results in every generator being weighted with $\frac{1}{3}$. We leave research on the effect of different distributions to future research, e.g., we can give more weight to adversarial attacks by assuming a higher probability of points near the boundaries.

4.4 Implementation

Alg. 3 implements an iteration of set-based training. First, a set-based forward propagation computes the output set $\widehat{\mathcal{Y}}$ for an ϵ-perturbance set (Lines 1–6). With $\widehat{\mathcal{Y}}$, the gradient of the set-based loss function $\mathcal{G}_\kappa$ is computed (Line 8). A set-based backpropagation computes the gradients $\mathcal{G}_k$ (Lines 9–13). Finally, the weights and biases of every linear layer are updated (Lines 14–17).

Algorithm 3: Set-based training iteration. Hyperparameters: $\epsilon \in \mathbb{R}_{>0}$, $\tau \in [0,1]$, and $\eta \in \mathbb{R}_{>0}$.

Data: Input $x \in \mathbb{R}^{n_0}$, Target $t \in \mathbb{R}^{n_\kappa}$
Result: Neural network with updated weights and biases

1 $\mathcal{H}_0 \leftarrow \langle x, \epsilon \, I_{n_0} \rangle_Z$ // Def. 2.11
2 **for** $k \leftarrow 1$ **to** κ **do** // set-based forward prop. (Prop. 2.12)
3 **if** *k-th layer is linear* **then**
4 $\mathcal{H}_k \leftarrow W_k \, \mathcal{H}_{k-1} + b_k$
5 **else**
6 $\mathcal{H}_k, \, m_k \leftarrow \texttt{fastEnclose}(L_k, \mathcal{H}_{k-1})$
7 $\widehat{\mathcal{Y}} \leftarrow \mathcal{H}_\kappa$
8 $\mathcal{G}_\kappa \leftarrow \frac{\partial \widetilde{E}(t, \widehat{\mathcal{Y}})}{\partial \widehat{\mathcal{Y}}}$ // Prop. 4.3
9 **for** $k \leftarrow \kappa$ **to** 1 **do** // set-based backprop. (Prop. 4.4)
10 **if** *k-th layer is linear* **then**
11 $\mathcal{G}_{k-1} \leftarrow W_k^\top \, \mathcal{G}_k$
12 **else**
13 $\mathcal{G}_{k-1} \leftarrow \mathrm{diag}(m_k) \, \mathcal{G}_k$
14 **for** $k \leftarrow 1$ **to** κ **do** // update weights and biases ((16))
15 **if** *k-th layer is linear* **then**
16 $W_k \leftarrow W_k - \eta \, \mathrm{E}[\mathbf{w}_k(\beta)]$
17 $b_k \leftarrow b_k - \eta \, \mathrm{E}[\mathbf{b}_k(\beta)]$

Proposition 4.8 (Time Complexity of Alg. 3). *Let $n_{max} \coloneqq \max_{k \in [\kappa]} n_k$ be the maximum number of neurons in a hidden layer of the neural network. The zonotopes used in Alg. 3 have at most $q \in O(n_0 + n_{max} \, \kappa)$ number of generators. Moreover, Alg. 3 has time complexity $O(n_{max}^2 \, q \, \kappa)$ w.r.t. n_{max}, q and the number of layers κ.*

Table 1: Networks and their number of parameters.

Dataset	Model	Hidden Neurons	Parameters
MNIST / F-MNIST	nn-small	2x100	89,610
	nn-med	5x100	119,910
	nn-large	7x250	575,260
SVHN	nn-med	5x100	348,710
	nn-large	7x250	1,147,260

Proof. The initial ϵ-perturbance set has n_0 generators (Line 1) and every activation layer adds $n_k \in O(n_{\max})$ new generators for the approximation errors (Alg. 2). Moreover, there are at most κ activation layers. Thus, in total, there are at most $\kappa\, O(n_{\max}) + O(n_0) = O(n_0 + n_{\max}\,\kappa)$ generators.

Time Complexity: The k-th step of the set-based forward propagation takes time $O(n_{\max}^2\, q)$: The linear map (Line 4) takes time $O(n_{k-1}\, n_k\, q) = O(n_{\max}^2\, q)$ (Prop. 2.9); the image enclosure (Line 6) takes time $O(n_{\max}^2\, q)$ (Prop. 3.7). Hence, the forward propagation (Lines 2–6) takes time $\kappa\, O(n_{\max}^2\, q) = O(n_{\max}^2\, q\,\kappa)$. Computing the gradient of the set-based loss function takes time $O(n_\kappa + n_\kappa\, q) = O(n_{\max}\, q)$. The k-th step of the set-based backpropagation computes a linear map (Line 11 or 13) that takes time $O(n_k\, n_{k+1}\, q) = O(n_{\max}^2\, q)$ (Prop. 2.9). Hence, the set-based backpropagation (Lines 9–13) takes time $\kappa\, O(n_{\max}^2\, q) = O(n_{\max}^2\, q\,\kappa)$. Updating a weight matrix takes time $O(n_{k+1}\, n_k + n_{k+1}\, n_k\, q) = O(n_{\max}^2\, q)$ (Line 16) and updating a bias vector takes time $O(n_k) = O(n_{\max})$ (Line 17). There are at most κ linear layers; hence, updating the weights and biases of all linear layers takes time $\kappa\,(O(n_{\max}^2\, q) + O(n_{\max})) = O(\kappa\, n_{\max}^2\, q)$. Thus, in total, an iteration of set-based training takes time $3\, O(n_{\max}^2\, q\,\kappa) = O(n_{\max}^2\, q\,\kappa)$. □

The time complexity of set-based training is polynomial, and compared to point-based training, only has an additional factor $q \in O(n_0 + n_{\max}\,\kappa)$. The increased time complexity is expected because set-based training propagates entire generator matrices through the neural network. Moreover, for some linear relaxation methods, similar time complexities are reported [45].

It is worth noting that set-based training only uses the operations: matrix-multiplication and matrix-addition, as well as min and max. Hence, a set-based training iteration can be efficiently evaluated for an entire batch of inputs using matrix operations on a GPU.

5 EVALUATION

We use the MATLAB toolbox CORA [1] to implement set-based training. The efficacy of set-based training is evaluated by training neural networks of three different sizes (see Tab. 1) on three different datasets: MNIST [24], Fashion-MNIST (F-MNIST) [42], and Street View House Numbers (SVHN) [30]. The training parameters can be found in the appendix. We compare set-based training (denoted by *set*) against point-based training (denoted by *point*) and training with adversarial inputs computed with PGD [27] (denoted by *PGD (training)*). We also evaluate set-based training ignoring the approximation errors (denoted by *set, no approx.*), which reduces the memory load. Moreover, we distinguish between the perturbation radius ϵ_{train} used during training and the perturbation radius

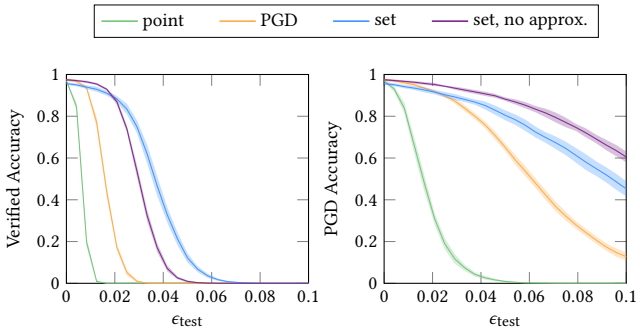

Figure 5: MNIST with nn-med and $\epsilon_{\text{train}} = 0.01$.

Table 2: SVHN (verified accuracy / PGD accuracy) [%].

		ϵ_{test}		
ϵ_{train}	**Method**	0.0	0.001	0.01
nn-med				
0.0	point	79.92	73.29 / 75.89	0.0 / 31.72
0.001	PGD	80.64	75.95 / 77.78	0.0 / 43.25
	set, no approx.	**80.98**	**78.78** / **79.16**	**0.12** / **61.36**
0.01	PGD	79.66	78.1 / 78.44	**0.1** / **62.71**
	set, no approx.	**80.06**	**78.22** / **78.66**	0.08 / 61.20
nn-large				
0.0	point	79.56	61.54 / 76.21	0.0 / 38.36
0.001	PGD	80.12	66.43 / 77.01	0.0 / 43.73
	set, no approx.	**84.67**	**82.24** / **83.19**	0.0 / **63.23**
0.01	PGD	76.01	72.99 / 74.39	0.0 / 57.68
	set, no approx.	**84.39**	**82.08** / **83.01**	0.0 / **63.99**

Table 3: MNIST (verified accuracy / PGD accuracy) [%].

		ϵ_{test}		
ϵ_{train}	**Method**	0.0	0.01	0.1
nn-small				
0.0	point	97.0	42.84 / 77.0	0.0 / 0.0
0.01	PGD	**97.84**	94.76 / **96.1**	0.0 / 12.9
	set	95.78	93.76 / 94.04	**0.7** / 53.12
	set, no approx.	97.12	**95.7** / 96.06	0.04 / **61.78**
	set, $\tau = 0$	95.4	93.26 / 93.62	0.14 / 35.36
0.1	PGD	**97.08**	**96.2** / **96.4**	0.08 / **84.3**
	set	72.24	67.98 / 69.09	**19.14** / 37.54
	set, no approx.	90.54	87.62 / 88.16	2.36 / 50.86
nn-med				
0.0	point	97.02	6.34 / 77.0	0.0 / 0.0
0.01	PGD	**97.6**	89.68 / 95.72	0.0 / 12.76
	set	95.62	93.54 / 93.84	0.0 / 45.24
	set, no approx.	97.46	**96.14** / **96.58**	0.0 / **60.52**
0.1	PGD	**97.48**	**96.6** / **96.84**	0.0 / **86.22**
	set	41.78	38.78 / 39.76	**9.42** / 21.18
	set, no approx.	90.52	86.95 / 87.7	0.68 / 46.6
nn-large				
0.0	point	97.94	0.0 / 89.28	0.0 / 0.02
0.01	PGD	**98.16**	4.36 / **96.82**	0.0 / 29.40
	set	96.56	**95.18** / 95.54	0.0 / **54.04**
0.1	PGD	**98.46**	**91.58** / **97.94**	0.0 / **87.32**
	set	33.66	31.6 / 32.64	**6.18** / 19.94
	set, no approx.	89.52	86.54 / 87.06	0.02 / 48.54

Table 4: F-MNIST (verified accuracy / PGD accuracy) [%].

		ϵ_{test}		
ϵ_{train}	**Method**	0.0	0.005	0.05
nn-small				
0.0	point	88.88	64.96 / 74.04	0.0 / 0.86
0.005	PGD	**89.02**	83.4 / **85.56**	0.0 / 33.2
	set	88.04	**84.12** / 84.98	0.0 / **50.12**
0.05	PGD	**83.32**	**82.32** / **82.6**	8.8 / **74.54**
	set	61.66	60.24 / 60.84	**42.36** / 55.48
nn-med				
0.0	point	88.08	19.2 / 70.66	0.0 / 1.32
0.005	PGD	88.12	72.68 / 84.78	0.0 / 28.1
	set	**88.38**	**82.94** / **85.2**	0.0 / **43.46**
0.05	PGD	**84.08**	**83.1** / **83.34**	0.0 / **75.28**
	set	54.92	53.8 / 54.56	**40.04** / 53.86

ϵ_{test} used during testing[1]. Ideally, we would like to report a neural network's adversarial accuracy for a perturbation radius. The adversarial accuracy is the minimum accuracy that can be achieved by perturbing the inputs of the test dataset. However, there is no efficient way to compute the adversarial accuracy [28]. Therefore, we report (i) the verified accuracy as a lower bound and (ii) the PGD accuracy as an upper bound for the adversarial accuracy. The verified accuracy is the percentage of test inputs for which we can formally verify the robustness of the neural network using set-based computations [23, Prop. B.2]. Moreover, the PGD accuracy is the accuracy of a neural network for adversarial inputs computed with PGD [22]. We list the findings of our experiments:

- For small training perturbation radii, the set-based trained neural networks achieve higher accuracies than the PGD-trained and point-based trained networks, e.g. for nn-med on F-MNIST with $\epsilon_{\text{train}} = 0.005$ (see Tab. 4) or for nn-large on SVHN with $\epsilon_{\text{train}} = 0.01$ (see Tab. 2).
- Set-based trained neural networks are significantly easier to verify, e.g., for nn-large on MNIST with $\epsilon_{\text{train}} = \epsilon_{\text{test}} = 0.01$, the verified accuracy of the set-based trained networks is 95.18% while the PGD trained networks and point-based trained networks achieve verified accuracies of 4.36% and 0.0% (see Tab. 3).
- Furthermore, set-based training with approximation errors trains neural networks with slightly lower clean accuracy ($\epsilon_{\text{test}} = 0$) and PGD accuracies than set-based training without approximation errors. However, set-based training with approximation errors achieves higher verified accuracies, which indicates that set-based training with approximation errors makes the trained neural networks easier to verify. Fig. 5 visualizes the accuracies.
- We observe that set-based training with the interval norm trains more accurate neural networks than set-based training without the interval norm (denoted by *set*, $\tau = 0$), e.g., see nn-small on MNIST with $\epsilon_{\text{train}} = 0.01$ (Tab. 3). This observation justifies the interval norm as a part of the set-based loss.

[1]All reported perturbation radii are w.r.t. normalized inputs between 0 and 1.

Table 5: Training time [s / epoch].

Dataset	Model	point	PGD	set	set, no approx.
MNIST	nn-small	3.87	13.5	8.91	7.81
	nn-med	6.17	22.07	17.39	13.35
	nn-large	8.69	31.51	65.43	29.48
F-MNIST	nn-small	4.54	13.11	9.42	8.01
	nn-med	7.52	20.50	17.77	13.47
SVHN	nn-med	10.18	26.33	-	74.14
	nn-large	12.78	34.23	-	141.17

- For larger training perturbation radii, the set-based trained networks achieve lower accuracies, e.g., clean accuracy ($\epsilon_{\text{test}} = 0$) of 72.24% for nn-small with $\epsilon_{\text{train}} = 0.1$ on MNIST (see Tab. 3). The reason for the lower accuracies is large approximation errors that build up during the set-based forward propagations. The large approximation errors distort the gradients and thereby prevent accurate training.

Set-based training on MNIST and F-MNIST with nn-small and nn-med is slower than point-based but slightly faster than PGD training (see Table 5). For nn-large on MNIST or SVHN, set-based training takes more time than point-based and PGD training.

Currently, the memory load and the large approximation errors limit the scalability of set-based training. The large approximation errors increase the outer approximation of the output sets, thereby increasing the number of false-positive outputs, i.e., outputs that are not true outputs. The false-positive outputs distort the gradients and prevent accurate training. Future research should explore methods to overcome both of these limitations. Furthermore, we only evaluate set-based training for classification tasks. However, without any modifications, set-based training can be applied to train neural networks for regression tasks: The gradient of the half-squared and cross-entropy errors are the same (Prop. 2.5).

6 CONCLUSION

We introduce the first end-to-end set-based training procedure for neural networks. While other robust training approaches train neural networks using single gradients and inputs, we use entire sets of gradients and inputs to train neural networks. We use set-based computations with zonotopes to efficiently compute sets of gradients for entire input sets. Finally, we update the parameters of neural networks by aggregating all gradients. Our experimental results demonstrate that our set-based approach effectively trains robust neural networks while significantly simplifying the subsequent robustness verification of the trained neural networks. In many instances, our set-based training approach outperforms training with state-of-the-art adversarial attacks. Hence, set-based training represents a promising new direction for the field of robust neural network training.

ACKNOWLEDGMENTS

The authors gratefully acknowledge partial financial support from the project SPP 2422 (No. 500936349) and the project FAI (No. 286525601), both funded by the German Research Foundation (DFG).

REFERENCES

[1] Matthias Althoff. 2015. An introduction to CORA 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems (ARCH)*, 120–151.

[2] Matthias Althoff. 2023. Checking and establishing reachset conformance in CORA 2023. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems (ARCH)*. Vol. 96, 9–33.

[3] Matthias Althoff. 2010. *Reachability analysis and its application to the safety assessment of autonomous cars*. Ph.D. Dissertation. Technische Universität München.

[4] Christopher M. Bishop. 2006. *Pattern recognition and machine learning*. Springer New York, NY.

[5] 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). *Int. Journal on Software Tools for Technology Transfer*, 25, 3, 329–339.

[6] Yinghao Chen, Dongdong Wang, Cao Kai, Cuijie Pan, Yayun Yu, and Muzhou Hou. 2022. Prediction of safety parameters of pressurized water reactor based on feature fusion neural network. *Ann. of Nuclear Energy*, 166.

[7] Christophe Combastel. 2015. Zonotopes and kalman observers: Gain optimality under distinct uncertainty paradigms and robust convergence. *Automatica*, 55, 265–273.

[8] Christophe Combastel and Ali Zolghadri. 2020. A distributed kalman filter with symbolic zonotopes and unique symbols provider for robust state estimation in cps. *Int. Journal of Control*, 93, 11, 2596–2612.

[9] Frederik Michel Dekking, Cornelis Kraaikamp, Hendrik Paul Lopuhaä, and Ludolf Erwin Meester. 2006. *A modern introduction to probability and statistics*. Springer London.

[10] György Elekes. 1986. A geometric inequality and the complexity of computing volume. *Discrete & Computational Geometry*, 1, 4, 289–292.

[11] Gidon Ernst et al. 2022. ARCH-COMP 2022 category report: Falsification with unbounded resources. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems (ARCH)*. Vol. 90, 204–221.

[12] 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 *IEEE Symposium on Security and Privacy (SP)*, 3–18.

[13] Antoine Girard. 2005. Reachability of uncertain linear systems using zonotopes. In *Proc. of the Int. Conf. on Hybrid Systems: Computation and Control (HSCC)*, 291–305.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT Press.

[15] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*.

[16] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Arthur Mann, and Pushmeet Kohli. 2019. Scalable verified training for provably robust image classification. In *Proc. of the IEEE/CVF Int. Conf. on Computer Vision (ICCV)*, 4841–4850.

[17] Geoffrey Hinton et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29, 6, 82–97.

[18] Ahmed Irfan, Kyle D. Julian, Haoze Wu, Clark Barrett, Mykel J. Kochenderfer, Baoluo Meng, and James Lopez. 2020. Towards verification of neural networks for small unmanned aircraft collision avoidance. In *AIAA/IEEE Digital Avionics Systems Conference (DASC)*, 1–10.

[19] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Int. Conf. on Computer Aided Verification (CAV)*, 97–117.

[20] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*.

[21] Niklas Kochdumper, Christian Schilling, Matthias Althoff, and Stanley Bak. 2023. Open- and closed-loop neural network verification using polynomial zonotopes. In *NASA Formal Methods*, 16–36.

[22] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2017. Adversarial machine learning at scale. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*.

[23] Tobias Ladner and Matthias Althoff. 2023. Automatic abstraction refinement in neural network verification using sensitivity analysis. In *Proc. of the Int. Conf. on Hybrid Systems: Computation and Control (HSCC)*, 1–13.

[24] Yann LeCun, Corinna Cortes, and CJ Burges. 2010. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2.

[25] Laura Lützow and Matthias Althoff. 2023. Reachability analysis of ARMAX models. In *Proc. of the IEEE Conf. on Decision and Control (CDC)*.

[26] Zhaoyang Lyu, Minghao Guo, Tong Wu, Guodong Xu, Kehuan Zhang, and Dahua Lin. 2021. Towards evaluating and training verifiably robust neural networks. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 4308–4317.

[27] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*.

[28] Matthew Mirman, Timon Gehr, and Martin Vechev. 2018. Differentiable abstract interpretation for provably robust neural networks. In *Proc. of the Int. Conf. on Machine Learning (ICML)*. Vol. 80, 3578–3586.

[29] Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. 2022. Prima: General and precise neural network certification via scalable convex hull approximations. *Proc. of the ACM on Programming Languages*, 6.

[30] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. 2011. Reading digits in natural images with unsupervised feature learning. In *Proc. of the NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.

[31] Eva Ostertagová. 2012. Modelling using polynomial regression. *Procedia Engineering*, 48, 500–506.

[32] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy (SP)*, 582–597.

[33] Andrew Ross and Finale Doshi-Velez. 2018. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In *Proc. of the AAAI Conf. on Artificial Intelligence (AAAI)*. Vol. 32.

[34] Bastian Schürmann and Matthias Althoff. 2017. Guaranteeing constraints of disturbed nonlinear systems using set-based optimal control in generator space. *IFAC-PapersOnLine*, 50, 1, 11515–11522.

[35] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. 2018. Fast and effective robustness certification. In *Proc. of the Int. Conf. on Neural Information Processing Systems (NeurIPS)*. Vol. 31.

[36] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin T. Vechev. 2019. Boosting robustness certification of neural networks. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*.

[37] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*.

[38] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. 2019. Robustness may be at odds with accuracy. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*.

[39] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. 2023. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 7464–7475.

[40] Lily Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. 2018. Towards fast computation of certified robustness for ReLU networks. In *Proc. of the Int. Conf. on Machine Learning (ICML)*. Vol. 80, 5276–5285.

[41] Eric Wong and Zico Kolter. 2018. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *Proc. of the Int. Conf. on Machine Learning (ICML)*. Vol. 80, 5286–5295.

[42] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747.

[43] Cunliang Ye, Yongfu Wang, Yunlong Wang, and Ming Tie. 2022. Steering angle prediction yolov5-based end-to-end adaptive neural network control for autonomous vehicles. *Proc. of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 236, 9, 1991–2011.

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

[45] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. 2018. Efficient neural network robustness certification with general activation functions. In *Proc. of the Int. Conf. on Neural Information Processing Systems (NeurIPS)*, 4944–4953.

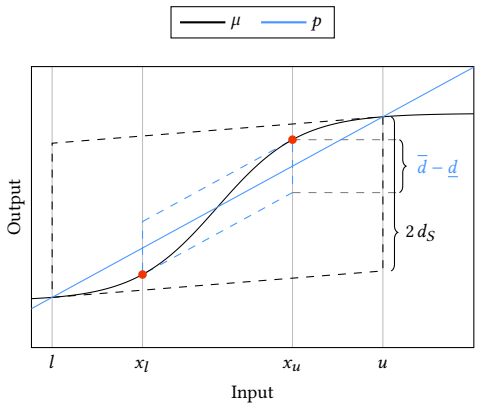

Figure 6: Illustration for Theorem 3.6.

A EVALUATION DETAILS

Our experiments were run on a server with 2×AMD EPYC 7763 (64 cores/128 threads), 2 TB RAM, and a NVIDIA A100 40GB GPU. We use the Adam optimizer [20] with the recommended hyperparameters. Furthermore, we use a batch size of 128 and train all networks for 100 epochs. Before training, we normalize all inputs between 0 and 1. All reported perturbation radii are with respect to the normalized inputs. Unless mentioned otherwise, we use the value $\tau = 0.0001$ for the set-based loss, which worked well in all our experiments.

Both MNIST and F-MNIST contain 60,000 grayscale images of size 28×28. Each image of MNIST depicts a handwritten digit from 0 to 9. An image of F-MNIST depicts a piece of clothing; there are ten categories, e.g., trousers, sneakers, or dress. SVHN is a real-world dataset that contains 73,257 colored images of digits of house numbers that are cropped to size 32×32. Each linear layer is followed by a nonlinear layer except for the last one. For MNIST, we use the ReLU activation function, while for F-MNIST and SVHN, we use the hyperbolic tangent. Each trained neural network is evaluated with a test dataset containing 500 unseen inputs. For point-based training, we use the MATLAB DeepLearning Toolbox[2]. The PGD attacks during training use $Q = 5$ iterations with step size $\epsilon_{\text{train}}/Q$. For PGD attacks during testing, we do $Q = 40$ iterations with step size ϵ_{test}/Q. Moreover, all of our reported accuracies are averaged over 10 runs.

Limitations of our Evaluation. The comparability of our evaluation results with other works is limited. Most other works use convolutional neural networks (CNN), whereas we only evaluate set-based training for feed-forward neural networks. Moreover, some other works use special input data normalization, e.g., [28], which changes the perturbation radius, preventing a meaningful comparison of accuracies.

B PROOFS

B.1 Section 3

Theorem 3.6. *Let $\mu : \mathbb{R} \to \mathbb{R}$ be an s-shaped function, and let $\mathcal{I} = [l, u] \in \mathbb{R}$ be an interval. Moreover, let $\underline{d}$ and $\overline{d}$ be the approximation errors of p as defined in Def. 3.1 and 3.2, and let d_S be the approximation error by [35, Thm. 3.2]. It holds that*

$$A([\underline{d}, \overline{d}], \mathcal{I}) \leq A([-d_S, d_S], \mathcal{I}).$$

Proof. We first observe that the approximation errors $\underline{d}$ and $\overline{d}$ of p can be computed at points $x_u, x_l \in \mathcal{I}$ such that $x_l \leq x_u$ (see Fig. 6):

$$\underline{d} = \mu(x_l) - p(x_l) \qquad \overline{d} = \mu(x_u) - p(x_u). \tag{18}$$

Moreover, the image of μ can be enclosed as follows [35, Thm. 3.2]:

$$\begin{aligned} m_S &= \min\left(\frac{\mathrm{d}\mu(l)}{\mathrm{d}l}, \frac{\mathrm{d}\mu(u)}{\mathrm{d}u}\right) \\ t_S &= \frac{1}{2}\left(\mu(u) + \mu(l) - m_S\,(u + l)\right) \\ d_S &= \frac{1}{2}\left(\mu(u) - \mu(l) - m_S\,(u - l)\right) \\ f_S(x) &= m_S\,x + t_S \end{aligned} \tag{19}$$

With Def. 3.1, we have the following inequality:

$$m_S \leq \frac{\mu(u) - \mu(l)}{u - l} = m. \tag{20}$$

Hence, we have

$$m\,(x_u - x_l) \geq m_S\,(x_u - x_l). \tag{21}$$

Moreover, from (19) we have for all $x \in \mathcal{I}$:

$$\begin{aligned} \mu(x) - \mu(l) \geq m_S\,(x - l) &\implies \mu(x_l) \geq \mu(l) + m_S\,(x_l - l), \\ \mu(u) - \mu(x) \geq m_S\,(u - x) &\implies \mu(x_u) \leq \mu(u) - m_S\,(u - x_u). \end{aligned} \tag{22}$$

Ultimately, we have

$$\begin{aligned} \overline{d} - \underline{d} &\overset{(18)}{=} \mu(x_u) - p(x_u) - (\mu(x_l) - p(x_l)) \\ &\overset{\text{Def. 3.1}}{=} \mu(x_u) - (m\,x_u + t) - (\mu(x_l) - (m\,x_l + t)) \\ &= \mu(x_u) - \mu(x_l) - m\,(x_u - x_l) \\ &\overset{(21)}{\leq} \mu(x_u) - \mu(x_l) - m_S\,(x_u - x_l) \\ &\overset{(22)}{\leq} \mu(u) - m_S\,(u - x_u) - (\mu(l) + \\ &\qquad m_S\,(x_l - l)) - m_S\,(x_u - x_l) \\ &= \mu(u) - \mu(l) - m_S\,(u - l) \\ &\overset{(19)}{=} 2\,d_S. \end{aligned} \tag{23}$$

Hence, we obtain the bound

$$\overline{d} - \underline{d} \leq 2\,d_S. \tag{24}$$

[2] https://de.mathworks.com/products/deep-learning.html

Thus,

$$\begin{aligned} A([\underline{d}, \overline{d}], \mathcal{I}) &\overset{(8)}{=} (u - l)\,(\overline{d} - \underline{d}) \\ &\overset{(24)}{\leq} (u - l)\,2\,d_S \\ &\overset{(8)}{=} A([-d_S, d_S], \mathcal{I}). \end{aligned} \tag{25}$$

□

B.2 Section 4

Proposition 4.4 (Set-Based Backpropagation). *Let $\widehat{\mathcal{Y}} \subset \mathbb{R}^{n_\kappa}$ be an output set of a neural network with target $t \in \mathbb{R}^{n_\kappa}$. The gradients $\mathcal{G}_k$ are computed in reverse order as*

$$\mathcal{G}_\kappa = \frac{\partial \widetilde{E}(t, \widehat{\mathcal{Y}})}{\partial \widehat{\mathcal{Y}}},$$

$$\mathcal{G}_{k-1} = \begin{cases} W_k^\top \mathcal{G}_k & \text{if } k\text{-th layer is linear,} \\ \operatorname{diag}(m_k)\, \mathcal{G}_k & \text{otherwise,} \end{cases}$$

for all $k \in \{\kappa, \dots, 1\}$. If the k-th layer is an activation layer, then the vector $m_k \in \mathbb{R}^{n_k}$ denotes the slopes of the linear approximations (Alg. 2).

Proof. If $k = \kappa$, we compute the gradient of the set-based loss according to Prop. 4.3. We assume $k < \kappa$. With an application of the chain rule to (11), we obtain the following:

$$\begin{aligned} \mathcal{G}_{k-1(i)} = \frac{\partial \widetilde{E}(t, \widehat{\mathcal{Y}})}{\partial \mathcal{H}_{k-1(i)}} &= \sum_{j=1}^{n_k} \frac{\partial \widetilde{E}(t, \widehat{\mathcal{Y}})}{\partial \mathcal{H}_{k(j)}} \frac{\partial \mathcal{H}_{k(j)}}{\partial \mathcal{H}_{k-1(i)}} \\ &= \sum_{j=1}^{n_k} \mathcal{G}_{k(j)} \frac{\partial \mathcal{H}_{k(j)}}{\partial \mathcal{H}_{k-1(i)}}. \end{aligned} \tag{26}$$

We split cases on the k-th layer type and simplify the terms.
Case (i). The k-th layer is a linear layer.

$$\begin{aligned} \mathcal{G}_{k-1(i)} &= \sum_{j=1}^{n_k} \mathcal{G}_{k(j)} \frac{\partial \mathcal{H}_{k(j)}}{\partial \mathcal{H}_{k-1(i)}} \\ &= \sum_{j=1}^{n_k} \mathcal{G}_{k(j)} \frac{\partial (W_{k(j,\cdot)}\, \mathcal{H}_k + b_{k-1(j)})}{\partial \mathcal{H}_{k-1(i)}} \\ &= \sum_{j=1}^{n_k} \mathcal{G}_{k(j)}\, W_{k(j,i)} \end{aligned} \tag{27}$$

Thus, $\mathcal{G}_{k-1} = W_k^\top \mathcal{G}_k$.
Case (ii). The k-th layer is an activation layer.

$$\begin{aligned} \mathcal{G}_{k-1(i)} = \sum_{j=1}^{n_k} \mathcal{G}_{k(j)} \frac{\partial \mathcal{H}_{k(j)}}{\partial \mathcal{H}_{k-1(i)}} &= \sum_{j=1}^{n_k} \mathcal{G}_{k(j)} \frac{\partial \mu_k(\mathcal{H}_{k-1(j)})}{\partial \mathcal{H}_{k-1(i)}} \\ &\overset{(12)}{\approx} \mathcal{G}_{k(i)}\, m_{k(i)} \end{aligned} \tag{28}$$

Thus, $\mathcal{G}_{k-1} = \operatorname{diag}(m_k)\, \mathcal{G}_k$.

□