Set-Based Training for Neural Network Verification

Lukas Koller, Tobias Ladner, and Matthias Althoff

***Abstract*—Neural networks are vulnerable to adversarial attacks, i.e., small input perturbations can significantly affect the outputs of a neural network. In safety-critical environments, the inputs often contain noisy sensor data; hence, in this case, neural networks that are robust against input perturbations are required. To ensure safety, the robustness of a neural network must be formally verified. However, training and formally verifying robust neural networks is challenging. We address both of these challenges by employing, for the first time, an end-to-end set-based training procedure that trains robust neural networks for formal verification. Our training procedure trains neural networks, which can be easily verified using simple polynomial-time verification algorithms. Moreover, our extensive evaluation demonstrates that our set-based training procedure effectively trains robust neural networks, which are easier to verify. Set-based trained neural networks consistently match or outperform those trained with state-of-the-art robust training approaches.**

***Index Terms*—Neural network verification, set-based computing, adversarial robustness, and adversarial training.**

I. Introduction

Neural networks demonstrate impressive performance for many complex tasks, such as speech recognition [1] or object detection [2]. However, many neural networks are sensitive to input perturbations [3]: Small, carefully chosen input perturbations can lead to vastly different outputs. This behavior is problematic for the adoption of neural networks in safety-critical environments, where the input often contains noisy sensor data or is subject to external disturbances, e.g., autonomous vehicle control [4] or airborne collision avoidance [5]. Thus, the formal verification of neural networks gained interest in recent years [6]. Given a set of inputs, the formal verification of neural networks attempts to find a proof that the neural network returns the correct output for every input from the set. Typically, when verifying the robustness of a neural network, the perturbations are modeled with the ℓ_∞-ball of radius $\epsilon \in \mathbb{R}_{>0}$ around an input. Subsequently, we provide a brief overview of related work.

A. Formal Verification of Neural Networks

The formal verification of neural networks is computationally challenging, i.e., with only rectified linear unit (ReLU) activation functions, it has been shown to be NP-hard [7]; thus, even verifying small neural networks often takes a long time. Most formal verification approaches either formulate the verification problem as an optimization problem or use reachability analysis [6]. Optimization-based approaches encode the verification problem as an optimization problem, which is solved using (mixed-integer) linear programming [8], [9] or satisfiability modulo theories (SMT) [7] solvers. Often, branch-and-bound algorithms [10], [11] are utilized for the verification of neural networks: The input space is recursively split until each subspace of the input space is either verified or falsified; due to the recursive process, branch-and-bound algorithms have a worst-case exponential runtime. Hence, many verification methods sacrifice accuracy for a polynomial runtime: Reachability analysis uses efficient set representations, e.g., zonotopes [12], combined with set-based computations to efficiently enclose the output set of a neural network (in polynomial time) [13], [14], [15], [16]. If the enclosure of the output set is sufficiently tight, it can be used to formally verify a neural network.

B. Adversarial Attacks

As an alternative to formal verification, neural networks can be falsified by adversarial attacks, i.e., small input perturbations that lead to an incorrect output. Often, adversarial attacks are fast to compute and effective at provoking incorrect outputs [17]. The most prominent approaches are the fast gradient sign method and projected gradient descent (PGD). The fast gradient sign method is a single-step gradient-based adversarial attack that efficiently generates adversarial attacks [17]. PGD uses multiple iterations of the fast gradient sign method to compute stronger adversarial attacks [18].

C. Training Robust Neural Networks

The training objective of a robust neural network is typically formulated as a min-max optimization problem [19]: minimize the worst-case loss within a set of input perturbances. Computing the worst-case loss within a set is computationally difficult [20]. Nonetheless, robust neural networks can be effectively trained by approximating the worst-case loss with adversarial attacks, e.g., computed with PGD [19].

Some approaches combine the training and formal verification of neural networks. In these works, the approximation of a worst-case loss is replaced by an upper bound, guaranteeing that no perturbation will lead to an incorrect output. Different methods for computing an upper bound of a worst-case loss within a set of perturbances have been proposed: Interval bound propagation (IBP) [21], linear relaxation [22], (mixed-integer) linear programming [23], or abstract interpretation [24]. IBP propagates input bounds through a neural network to obtain conservative output bounds; the worst-case output within the computed output bounds can be used for training and verification [21]. State-of-the-art robustness results are achieved by combining IBP of small regions with adversarial attacks [25]; however, a branch-and-bound algorithm with worst-case exponential-time complexity is used for their formal verification. We aim to train neural networks that can be verified using simple polynomial-time verification algorithms. More closely related to this work is an approach

Preprint (under review)

using abstract interpretation [24], which also utilizes set-based computations with zonotopes to compute an outer approximation of the output set during training; however, much set-based information is discarded by only using the output set to bound a worst-case loss. Conversely, our approach computes a loss for the entire output set. Moreover, our training approach can be viewed as a set-based extension of the well-established tradeoff-loss [26], which combines a regular loss for accuracy with a boundary loss for robustness. The boundary loss pushes the decision boundary of a classifier away from the training samples and thereby increases the robustness of the trained neural network.

Furthermore, the training of robust neural networks can be improved by using a special initialization for the weights and biases [27]. Other approaches for training robust neural networks include input gradient regularization [28], where large gradients are penalized, or neural network destillation [29], which trains a second neural network with the predictions of a first one trained with the training data.

D. Contributions

Our main contributions are:

- The first end-to-end set-based training procedure for robust neural networks that can be easily verified using simple polynomial-time verification algorithms. Our training procedure has only two additional hyperparameters compared to standard neural network training.
- A set-based loss function that extends the well-known tradeoff-loss [26], which computes a loss for an entire output set.
- Our image enclosure of nonlinear layers using linear approximations that can be efficiently computed for an entire batch of input sets using matrix operations on a GPU.
- We prove that the approximation errors by our image enclosure are always smaller or equal compared to Singh's enclosure [14, Thm. 3.2].
- An extensive empirical evaluation of our set-based training approach with neural networks of various sizes trained on different datasets. We demonstrate the efficacy of set-based training and that set-based trained neural networks match or outperform neural networks trained with state-of-the-art robust training approaches.

E. Organization

We introduce the required preliminaries in Sec. II. An efficient image enclosure is derived in Sec. III that we use for our set-based training procedure, introduced in Sec. IV. We provide an empirical evaluation in Sec. V. Finally, we conclude our findings in Sec. VI.

II. Preliminaries

A. 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 $A \in \mathbb{R}^{n\times m}$, $A_{(i,j)}$ 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. Given two matrices $A \in \mathbb{R}^{m\times n_1}$ and $B \in \mathbb{R}^{m\times n_2}$, their (horizontal) concatenation is denoted by $\left[\,A\;B\,\right] \in \mathbb{R}^{m\times(n_1+n_2)}$; if $n_1 = n_2$, their Hadamard product is the element-wise multiplication $(A \odot B)_{(i,j)} = A_{(i,j)}\,B_{(i,j)}$. The operation $\mathrm{Diag}\colon \mathbb{R}^n \to \mathbb{R}^{n\times n}$ returns a diagonal matrix with the entries of a given vector on its diagonal; its counterpart is the operation $\mathrm{diag}\colon \mathbb{R}^{n\times n} \to \mathbb{R}^n$, which returns a vector which contains the diagonal entries of a given square 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 by $\mathcal{S}_{(i)}$. Given two sets $\mathcal{S}_1 \subset \mathbb{R}^n$ and $\mathcal{S}_2 \subset \mathbb{R}^m$, we denote the Cartesian product by $\mathcal{S}_1 \times \mathcal{S}_2 = \left\{\left[\,s_1^\top\; s_2^\top\,\right]^\top \mid s_1 \in \mathcal{S}_1,\, s_2 \in \mathcal{S}_2\right\}$, 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 $n \in \mathbb{N}$, $[n] = \{1, 2, \dots, n\}$ denotes the set of all natural numbers up to n. An n-dimensional interval $\mathcal{I} \subset \mathbb{R}^n$ with bounds $l, u \in \mathbb{R}^n$ is denoted by $\mathcal{I} = [l, u]$, where $\forall i \in [n]\colon l_{(i)} \leq u_{(i)}$. For a function $f\colon \mathbb{R}^n \to \mathbb{R}^m$, we abbreviate its evaluation for a set $\mathcal{S} \subset \mathbb{R}^n$ with $f(\mathcal{S}) = \{f(s) \mid s \in \mathcal{S}\}$. The derivative of a scalar function $f\colon \mathbb{R} \to \mathbb{R}$ is denoted as $f'(x) = {}^{\mathrm{d}}\!/\!_{\mathrm{d}x}\, f(x)$. Moreover, the gradient of a function $f\colon \mathbb{R}^n \to \mathbb{R}$ w.r.t. a vector $x \in \mathbb{R}^n$ is its element-wise derivative: $(\nabla_x f(x))_{(i)} = {}^{\partial}\!/\!_{\partial x_{(i)}}\, f(x)$, for $i \in [n]$. Analogously, we define the gradient of a function $f\colon \mathbb{R}^{n\times m} \to \mathbb{R}$ w.r.t. a matrix $A \in \mathbb{R}^{n\times m}$: $(\nabla_A f(A))_{(i,j)} = {}^{\partial}\!/\!_{\partial A_{(i,j)}}\, f(A)$, for $i \in [n]$ and $j \in [m]$.

B. 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 a nonlinear (activation) layer, which applies a nonlinear activation function element-wise.

Definition 1 (Neural Network Layer, [30, Sec. 5.1])**.** *For the k-th layer, $n_{k-1} \in \mathbb{N}$ denotes the number of input neurons and $n_k \in \mathbb{N}$ denotes the number of output neurons; if the k-th layer is linear, there is a weight matrix $W_k \in \mathbb{R}^{n_k\times n_{k-1}}$ and a bias vector $b_k \in \mathbb{R}^{n_k}$, otherwise there is nonlinear activation function $\mu_k(\cdot)$ which is applied element-wise. The k-th layer is defined as an operation $L_k\colon \mathbb{R}^{n_{k-1}} \to \mathbb{R}^{n_k}$,*

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

With θ, we denote the parameters of the neural network, which include all weight matrices and bias vectors from its linear layers.

Definition 2 (Forward Propagation, [30, 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*

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

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

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

Definition 3 (Cross-Entropy Error, [30, Sec. 5.2])**.** *The cross-entropy error $E_{CE} \colon \mathbb{R}^{n_\kappa} \times \mathbb{R}^{n_\kappa} \to \mathbb{R}$ is defined as*

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

where $\ln(\cdot)$ denotes the natural logarithm and $p_{(i)} = \exp(y_{(i)})/(\exp(y)\,\mathbf{1})$ are the predicted class probabilities.

The training goal of a neural network is to find network parameters θ that minimize the total loss of the training dataset $\mathcal{D}$ [30, 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 [30, Sec. 5.2.4]: the parameters are iteratively optimized using the gradient of the loss function. The parameters are initialized randomly, e.g., see [31]. Let us introduce the gradient g_k of the loss function E w.r.t. the output of the k-th layer h_k for the input $x \in \mathbb{R}^{n_0}$:

$$g_k \coloneqq \nabla_{h_k} E(t, y), \tag{2}$$

where $y = N_\theta(x)$. The weight matrix W_k and bias vector b_k of the k-th layer are updated as [30, Sec. 5.3]

$$\begin{aligned} W_k &\leftarrow W_k - \eta\, \nabla_{W_k} E(t, y) = W_k - \eta\, g_k\, h_{k-1}^\top, \\ b_k &\leftarrow b_k - \eta\, \nabla_{b_k} E(t, y) = b_k - \eta\, g_k, \end{aligned} \tag{3}$$

where $\eta \in \mathbb{R}_{>0}$ is the learning rate. The gradients g_k are efficiently computed with backpropagation [30, Sec. 5.3]: by utilizing the chain rule, the gradient g_κ of the last layer is propagated backward through all neural network layers.

Proposition 1 (Backpropagation, [30, Sec. 5.3])**.** *Let $y \in \mathbb{R}^{n_\kappa}$ be an output of a neural network with target $t \in \mathbb{R}^{n_\kappa}$. The gradients g_k are computed in reverse order as*

$$\begin{aligned} g_\kappa &= \nabla_y E(t, y), \\ g_{k-1} &= \begin{cases} W_k^\top\, g_k & \text{if } k\text{-th layer is linear}, \\ \operatorname{Diag}\big(\mu_k'(h_{k-1})\big)\, g_k & \text{otherwise}, \end{cases} \end{aligned}$$

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

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

C. Set-Based Computation

Our approach extends point-based training to sets, which we represent with zonotopes. A zonotope is a convex set representation describing the Minkowski sum of a finite number of line segments.

Definition 4 (Zonotope, [12, 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*

$$\mathcal{Z} = \{\, c + G\,\beta \mid \beta \in [-1, 1]^q \,\} \eqqcolon \langle c, G \rangle_Z.$$

Subsequently, we define several operations for zonotopes used in our training approach.

Proposition 2 (Interval Enclosure, [32, 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*

$$l = c - |G|\,\mathbf{1}, \qquad u = c + |G|\,\mathbf{1},$$

where $|\cdot|$ computes the element-wise absolute value. The time complexity of computing an interval enclosure is $\mathcal{O}(n\,q)$.

Proposition 3 (Minkowski Sum, [32, 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*

$$\mathcal{Z} \oplus \mathcal{I} = \big\langle c + {}^1\!/\!_2\,(u + l), \big[G \quad {}^1\!/\!_2\, \operatorname{Diag}(u - l)\big] \big\rangle_Z,$$

and has time complexity $\mathcal{O}(n\,(n + q))$.

Proposition 4 (Affine Map, [32, Sec. 2.4])**.** *The result of an affine map $f \colon \mathbb{R}^n \to \mathbb{R}^m$, $x \mapsto W\,x + b$ 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 $\mathcal{O}(m\,n\,q)$.

Determining the volume of a zonotope is computationally demanding [33]. However, we can effectively approximate the size of a zonotope with its F-radius [34]: The F-radius of a zonotope is the Frobenius norm of its generator matrix.

Proposition 5 (F-Radius, [34, Def. 3])**.** *For a zonotope $\mathcal{Z} = \langle c, G \rangle_Z \subset \mathbb{R}^n$ with $G \in \mathbb{R}^{n \times q}$, the F-radius is*

$$\|\mathcal{Z}\|_F \coloneqq {}^1\!/\!_n \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{q} G_{(i,j)}^2} = {}^1\!/\!_n \sqrt{\mathbf{1}^\top (G \odot G)\,\mathbf{1}},$$

where $G \in \mathbb{R}^{n \times q}$.

D. 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. The target output $t \in \mathbb{R}^{n_\kappa}$ is a one-hot encoding of the target label $l \in [n_\kappa]$, i.e. $t = e_l$ is the l-th standard basis vector e_l. An input $x \in \mathbb{R}^{n_0}$ is correctly classified by a neural

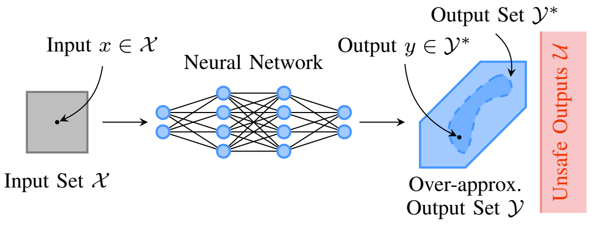
Input $x \in \mathcal{X}$
Neural Network
Output $y \in \mathcal{Y}^*$
Output Set $\mathcal{Y}^*$
Unsafe Outputs $\mathcal{U}$
Input Set $\mathcal{X}$
Over-approx.
Output Set $\mathcal{Y}$

Fig. 1. Verifying the local robustness of a neural network.

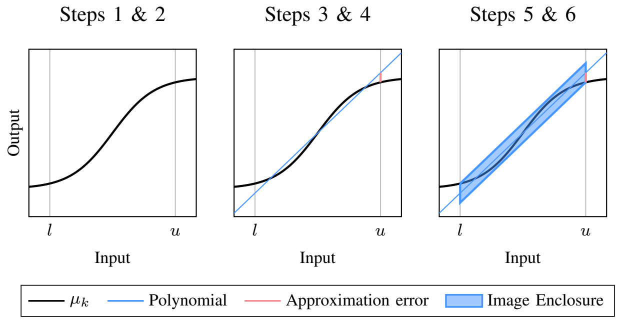

Fig. 2. Main steps of an image enclosure [16, Prop. 2.14].

network if the predicted classification label matches the target label:

$$\arg\max_{k\in[n_\kappa]} y_{(k)} = l. \tag{4}$$

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

$$\pi_\epsilon(x) := \langle x, \epsilon\, I_{n_0} \rangle_Z = \{\, \tilde{x} \in \mathbb{R}^{n_0} \mid \|\tilde{x} - x\|_\infty \le \epsilon \}. \tag{5}$$

For an input set $\mathcal{X} = \pi_\epsilon(x) \subset \mathbb{R}^{n_0}$, we formally verify the robustness of a neural network N_θ by using set-based computations to efficiently compute an outer approximation $\mathcal{Y} \subset \mathbb{R}^{n_\kappa}$ of its output set $\mathcal{Y}^* := N_\theta(\mathcal{X}) \subseteq \mathcal{Y}$ (in polynomial time). If $\mathcal{Y}$ does not intersect with a region of unsafe outputs $\mathcal{U}$, we have formally verified the neural network for the input set $\mathcal{X}$, as also $\mathcal{Y}^*$ does not intersect with $\mathcal{U}$. Fig. 1 illustrates the formal verification of a neural network. For a classification task with target label $l \in \mathbb{R}^{n_\kappa}$, the unsafe set contains every incorrect classification [16, Prop. B.2], i.e. there is a dimension $k \in [n_\kappa]$ for which the output $y_{(k)}$ is larger than the output of the target dimension $y_{(l)}$:

$$\mathcal{U}_t := \left\{ y \in \mathbb{R}^{n_\kappa} \;\middle|\; \exists k \in [n_\kappa] \colon y_{(k)} > y_{(l)} \right\}. \tag{6}$$

To compute an outer approximation $\mathcal{Y}$, we evaluate the operations L_k (Def. 1) by sets. We can exactly compute the output set of a linear layer for an input set with a linear map [16, Sec. 2.4]. However, the output set of a nonlinear layer is enclosed as it cannot be computed exactly for zonotopes, i.e., zonotopes are not closed under nonlinear maps. The required steps are summarized in Fig. 2. The activation function is applied element-wise; hence, the input dimensions are considered independently. We first compute an upper and lower bound for each dimension of the input set (steps 1 & 2). The activation function is approximated within the computed bounds using a linear polynomial (step 3). To ensure the soundness of the approximations, a bound on the approximation errors is computed (Step 4) and added to the result (steps 5 & 6).

We define the following set-based forward propagation.

Proposition 6 (Set-Based Forward Prop., [16, Sec. 2.4])**.** *For an input set $\mathcal{X} \subset \mathbb{R}^{n_0}$, an outer approximation $\mathcal{Y} \subset \mathbb{R}^{n_\kappa}$ of the output set $\mathcal{Y}^* := N_\theta(\mathcal{X})$ of a neural network can be computed as*

$$\begin{aligned} \mathcal{H}_0 &= \mathcal{X}, \\ \mathcal{H}_k &= L_k(\mathcal{H}_{k-1}) \quad \textit{for } k \in [\kappa], \\ \mathcal{Y} &= \mathcal{H}_\kappa, \end{aligned}$$

where for each $k \in [\kappa]$,

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

and the operation `enclose` *computes an image enclosure of a nonlinear layer, e.g. [16, Prop. 2.14]. Moreover, let $\mathcal{H}_k = \langle c_k, G_k \rangle_Z$ for $k \in \{0, 1, \dots, \kappa\}$.*

E. Problem Statement

The training goal for a robust neural network is to minimize the worst-case loss within the ℓ_∞-ball of radius $\epsilon \in \mathbb{R}_{>0}$ around each training input [19, 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 want to derive a set-based training procedure that uses set-based computations to train robust neural networks, which can be verified with simple polynomial-time verification algorithms.

III. Fast, Batch-wise Image Enclosure of Activation Functions

The training of neural networks requires many forward propagations. Hence, we want to efficiently compute image enclosures for an entire batch of input sets. Therefore, sampling-based methods [16], [15], which use evenly distributed samples along the activation function to bound the approximation errors (see Fig. 2), for the image enclosure are impractical. In contrast, [14] derives fast analytical solutions for the approximation errors of a specific linear approximation of s-shaped activation functions. However, these linear approximations create rather 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, hyperbolic tangent, and logistic sigmoid. Secondly, we

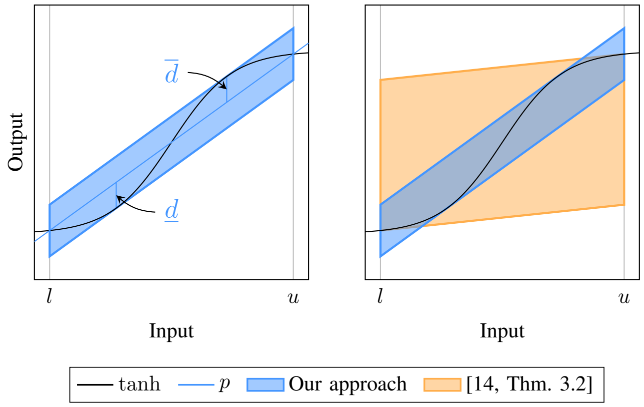

Fig. 3. Image enclosure of hyperbolic tangent: (left) Our linear approximation and approximation errors; (right) Comparison of our image enclosure and Singh's enclosure [14, Thm. 3.2].

provide a linear approximation whose approximation errors are smaller or equal to Singh's enclosure [14, Thm. 3.2] while being equally fast to compute.

For the remainder of this section, let $\mu \colon \mathbb{R} \to \mathbb{R}$ be a monotonically increasing function, which is approximated by a linear function $p \colon \mathbb{R} \to \mathbb{R}$, $x \mapsto a\,x + b$ within an interval $[l, u] \subset \mathbb{R}$. The approximation errors of p are given by the largest lower distance and upper distance between μ and p (see Fig. 3).

Definition 5 (Approximation Error of Linear Approximation)**.** *The approximation errors of a linear approximation p for μ within the interval $[l, u]$ are defined as*

$$\underline{d} \coloneqq \min_{x \in [l,u]} \mu(x) - p(x), \qquad \overline{d} \coloneqq \max_{x \in [l,u]} \mu(x) - p(x).$$

Using the approximation errors, we can enclose the output of μ,

$$\forall x \in [l, u] \colon \mu(x) \in p(x) + d_c \oplus [-d, d],$$

where $d_c = {}^1\!/_2\,(\overline{d} + \underline{d})$ and $d = {}^1\!/_2\,(\overline{d} - \underline{d})$. Fig. 3 illustrates a linear approximation p of the hyperbolic tangent along with its approximation errors. We efficiently find the approximation errors $\underline{d}$ and $\overline{d}$ of a linear approximation p by only evaluating its difference with the activation $\mu(x) - p(x)$ at a finite subset of points of the interval $[l, u]$.

A. Efficient Computation of Approximation Errors

The rectified linear unit (ReLU) is piece-wise linear and defined as $\mathrm{ReLU}(x) \coloneqq \max(0, x)$.

Proposition 7 (Approximation Errors for ReLU)**.** *The approximation errors of p for ReLU are computed as*

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

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

Proof. $\mathrm{ReLU}(x) - p(x)$ is linear for $[l, 0]$ and $[0, u]$. Thus, the approximation errors are found at the bounds $x \in \{l, u\}$ or where $0 \in [l, u]$ at $x = 0$. □

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.

Proposition 8 (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 $\mathcal{P} = \left\{\pm \tanh^{-1}\!\left(\sqrt{1 - a}\right), l, u\right\} \cap [l, u]$.

Proof. The derivative of the hyperbolic tangent is $\tanh'(x) = 1 - \tanh(x)^2$. To compute the extreme points of $\tanh(x) - p(x)$, we demand that its derivative is 0 and simplify the terms:

$$\begin{aligned} & 0 \overset{!}{=} {}^{\mathrm{d}}\!/_{\mathrm{d}x}(\tanh(x) - p(x)) \\ \Leftrightarrow \quad & 0 = 1 - \tanh(x)^2 - a \\ \Leftrightarrow \quad & \tanh(x) = \pm\sqrt{1 - a} \\ \Leftrightarrow \quad & x = \pm \tanh^{-1}\!\left(\sqrt{1 - a}\right). \end{aligned}$$ □

Proposition 9 (Approximation Errors for Logistic Sigmoid)**.** *The approximation errors of p for $\sigma(x) = {}^1\!/_2\,(\tanh({}^x\!/_2) + 1)$ [35, Sec. 6.3.2] 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 $\mathcal{P} = \left\{\pm 2\, \tanh^{-1}\!\left(\sqrt{1 - 4\,a}\right), l, u\right\} \cap [l, u]$.

Proof. To compute the extreme points of $\sigma(x) - p(x)$, we demand that its derivative is 0 and simplify the terms:

$$\begin{aligned} & 0 \overset{!}{=} {}^{\mathrm{d}}\!/_{\mathrm{d}x}(\sigma(x) - p(x)) \\ \Leftrightarrow \quad & 0 = {}^{\mathrm{d}}\!/_{\mathrm{d}x}({}^1\!/_2\,(\tanh({}^x\!/_2) + 1) - p(x)) \\ \Leftrightarrow \quad & 0 = {}^1\!/_2 \left(1 - \tanh({}^x\!/_2)^2\right) {}^1\!/_2 - a \\ \Leftrightarrow \quad & 0 = 1 - \tanh({}^x\!/_2)^2 - 4\,a \\ \Leftrightarrow \quad & \tanh({}^x\!/_2) = \pm\sqrt{1 - 4\,a} \\ \Leftrightarrow \quad & x = \pm 2\, \tanh^{-1}\!\left(\sqrt{1 - 4\,a}\right). \end{aligned}$$ □

We note that our computation of the approximation errors works for any (monotonically increasing) linear approximation. Moreover, we observe that the offset b of the linear approximation p has no effect on the image enclosure; hence, w.l.o.g. we set $b = 0$.

Definition 6 (Linear Approximation of an Activation Function)**.** *Within the interval $[l, u]$, we approximate μ by a linear function $p(x) \coloneqq a\,x$, where*

$$a \coloneqq \frac{\mu(u) - \mu(l)}{u - l}.$$

B. Our Enclosure vs. Singh's Enclosure

For s-shaped activation functions, e.g., hyperbolic tangent and logistic sigmoid, we prove that the approximation errors of our linear approximation (Def. 6) are always smaller or equal to the approximation errors of Singh's enclosure [14, Thm. 3.2] w.r.t. the area in the input-output plane (see Fig. 3) measuring the integrated approximation error over $[l, u]$:

$$\mathrm{area}([\underline{d}, \overline{d}], [l, u]) \coloneqq (u - l)\,(\overline{d} - \underline{d}). \tag{8}$$

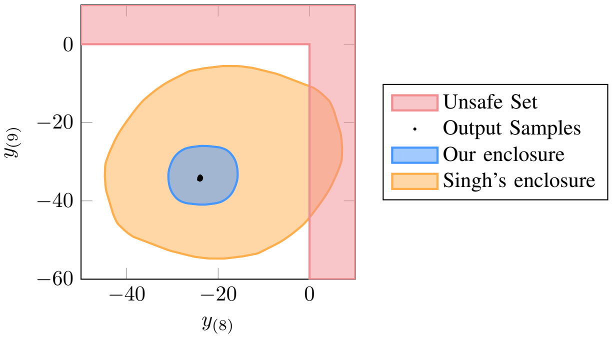

Fig. 4. Comparison of the output set of a neural network computed with our image enclosures and Singh's enclosure [14, Thm. 3.2]. The details for reproducibility can be found in the appendix.

Theorem 1. *Let μ be an s-shaped function, and let $[l, u]$ be an interval. Moreover, let $\underline{d}$ and $\overline{d}$ be the approximation errors of p as defined in Def. 5 and 6, and let d_S be the approximation error of Singh's enclosure [14, Thm. 3.2]. It holds that*

$$\operatorname{area}([\underline{d}, \overline{d}], [l, u]) \leq \operatorname{area}([-d_S, d_S], [l, u]).$$

Proof. See appendix. □

Fig. 4 shows an instance where the smaller approximation errors by our image enclosure enable the verification of a neural network that is not possible with Singh's enclosure [14, Thm. 3.2]: The output set computed with our image enclosure is significantly smaller and does not intersect the unsafe region, while the output set computed with Singh's enclosure does intersect the unsafe region.

C. Computational Complexity

Alg. 1 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 8), and the approximation errors (Line 9) are added. The time complexity of Alg. 1 is polynomial w.r.t. the number of input dimensions and the number of generators.

Proposition 10 (Time Complexity of Alg. 1)**.** *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. 1 has time complexity $\mathcal{O}(n^2\, q)$ 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 $\mathcal{O}(n\, q)$ (Prop. 2). Computing the linear approximation (Line 4) and the approximation errors (Line 5) for each neuron takes constant time; hence, the loop takes time $\mathcal{O}(n)$. The linear map of $\mathcal{H}_{k-1}$ (Line 8) takes time $\mathcal{O}(n^2\, q)$ (Prop. 4). Adding the approximation errors (Line 9) takes time $\mathcal{O}(n\,(n+q))$. Thus, in total we have $\mathcal{O}(n\, q) + \mathcal{O}(n) + \mathcal{O}(n^2\, q) + \mathcal{O}(n\,(n+q)) = \mathcal{O}(n^2\, q)$. □

Alg. 1 no longer uses a polynomial regression or requires sampling to compute the approximation errors [15, Sec. 3.2]. Moreover, each loop iteration of Alg. 1 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.

Algorithm 1: Fast image enclosure of a nonlinear layer.

```
1  function fastEnc($\mu_k$, $\mathcal{H}_{k-1}$)
2      Find bounds $[l_{k-1}, u_{k-1}]$ of $\mathcal{H}_{k-1}$                // Prop. 2
3      for $i \leftarrow 1$ to $n_k$ do
4          Find linear approx. $a_{k(i)}\, x$ of $\mu_k$                     // Def. 6
5          Find approx. errors $\underline{d}_{k(i)}$, $\overline{d}_{k(i)}$          // Prop. 7 to 9
6          $d_{c,k(i)} \leftarrow {}^{1}/_{2}\,(\overline{d}_{k(i)} + \underline{d}_{k(i)})$        // Def. 5
7          $d_{k(i)} \leftarrow {}^{1}/_{2}\,(\overline{d}_{k(i)} - \underline{d}_{k(i)})$          // Def. 5
8      $\widetilde{\mathcal{H}}_k \leftarrow \operatorname{Diag}(a_k)\, \mathcal{H}_{k-1} + d_{c,k}$       // Prop. 4
9      $\mathcal{H}_k \leftarrow \widetilde{\mathcal{H}}_k \oplus [-d_k, d_k]$                   // Prop. 3
10     return $\mathcal{H}_k$
```

IV. 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-based loss using the entire output set of an ϵ-perturbance set (5), (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.

A. Set-Based Loss

First, we define a set-based loss function returning a loss for an entire set of outputs. In our work, we define a set-based loss function $\widetilde{E} : \mathbb{R}^{n_\kappa} \times 2^{\mathbb{R}^{n_\kappa}} \to \mathbb{R}^{n_\kappa}$ so that it combines (i) the point-based loss of the center of the output set with (ii) the F-radius of the output set (Prop. 5). The included F-radius minimizes the size of the output sets, thereby increasing the robustness of the trained neural network, while the point-based loss of the center trains the accuracy.

Definition 7 (Set-Based Loss)**.** *Given a (point-based) loss function $E \colon \mathbb{R}^{n_\kappa} \times \mathbb{R}^{n_\kappa} \to \mathbb{R}$, we define a set-based loss function as*

$$\widetilde{E}(t, \mathcal{Y}) := (1 - \tau)\, E(t, c_\kappa) + {}^{\tau}/_{\epsilon}\, \|\mathcal{Y}\|_F,$$

where $\epsilon \in \mathbb{R}_{>0}$ is the training perturbation radius and $\mathcal{Y} = \langle c_\kappa, G_\kappa \rangle_Z$ is an output set.

The set-based loss function balances the point-based center loss and the F-radius using a hyperparameter $\tau \in [0, 1]$. To make tuning the hyperparameter τ easier, the F-radius in Def. 7 is normalized with the input perturbation radius $\epsilon \in \mathbb{R}_{>0}$. The normalization is derived from the F-radius of the input set $\mathcal{X} = \pi_\epsilon(x)$ for an input x:

$$\|\pi_\epsilon(x)\|_F \overset{(5)}{=} \epsilon. \tag{9}$$

The set-based loss can be viewed as a set-based extension of the well-established tradeoff-loss [26, Eq. 5], which combines

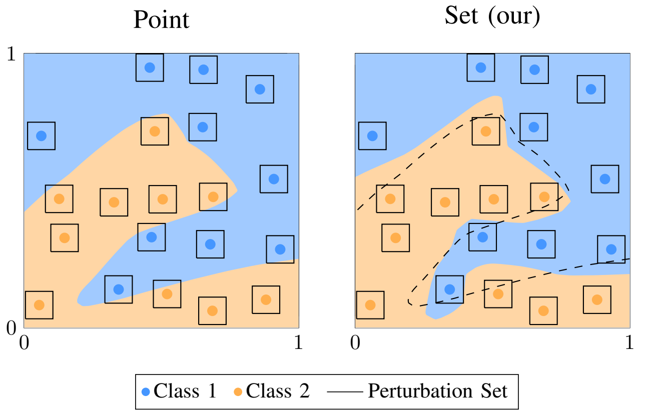

Fig. 5. Comparing the decision bounds of point-based (left) and set-based (right) training. The dashed line is the decision boundary of point-based training. The details for reproducibility can be found in the appendix.

a standard training loss (first summand) with a boundary loss (second summand):

$$E_{\text{TRADES}}(t,y) = E(t,y) + \max_{\tilde{x}\in\pi_\epsilon(x)} {}^1\!/\!_\lambda\, E(y, N_\theta(\tilde{x})),$$

where $y = N_\theta(x)$ and λ is a weighting factor. removedThe maximization of the boundary loss is approximated using PGD. The boundary loss smooths the output of a neural network by pushing the decision boundaries away from the training samples. Similar to the F-radius, the boundary loss measures the size of the output set. However, the F-radius captures the size of an output set in all dimensions, while the boundary loss only considers the size w.r.t. one direction. Moreover, through the sound set-based computations (Prop. 6), the set-based loss accurately over-approximates the size of the output set. In contrast, the boundary loss is only approximated in [26].

Fig. 5 compares the learned decision boundaries of point-based and set-based training for a simple binary classification task. Both training methods perfectly learn the training data, but we can see that set-based training pushes the decision boundaries away from the samples: For some samples, the decision boundary of the point-based trained neural network crosses their perturbation sets, which is not the case for the set-based trained neural network. Thus, the set-based trained model is more robust.

B. Set-Based Backpropagation

In this section, we now lift the point-based backpropagation (Prop. 1) to a set-based evaluation. We first derive the gradient of the set-based loss function w.r.t. the output set $\mathcal{Y}$, which is also 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.

Definition 8 (Zonotope Gradient)**.** *The gradient of a function $f(\cdot)$ w.r.t. a zonotope $\mathcal{Z} = \langle c, G\rangle_Z \subset \mathbb{R}^n$ is defined as*

$$\nabla_{\mathcal{Z}} f(\mathcal{Z}) := \langle \nabla_c f(\mathcal{Z}), \nabla_G f(\mathcal{Z})\rangle_Z.$$

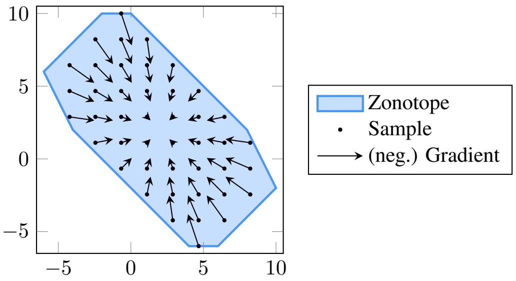

Fig. 6. Gradients of the F-radius of a zonotope.

The second term of the set-based loss function is the F-radius of the output set, the gradient of which is computed as follows.

Proposition 11 (Gradient of F-Radius)**.** *The gradient of the F-radius is*

$$\nabla_{\mathcal{Y}} \|\mathcal{Y}\|_F = \frac{1}{n_\kappa \|\mathcal{Y}\|_F} \langle \mathbf{0}, G_\kappa\rangle_Z,$$

where $\mathcal{Y} = \langle c_\kappa, G_\kappa\rangle_Z \subset \mathbb{R}^n_\kappa$.

Proof. The center does not affect the F-radius, hence $\nabla_{c_\kappa}\|\mathcal{Y}\|_F = \mathbf{0}$. The F-radius is the sum of all squared entries of the generator matrix. Hence,

$$\nabla_{G_\kappa}\|\mathcal{Y}\|_F = \frac{1}{n_\kappa}\nabla_{G_\kappa}\sqrt{\mathbf{1}^\top (G_\kappa \odot G_\kappa)\,\mathbf{1}} = \frac{G_\kappa}{n_\kappa\|\mathcal{Y}\|_F}. \qquad (10)$$

Thus,

$$\begin{aligned}\nabla_{\mathcal{Y}}\|\mathcal{Y}\|_F &\overset{\text{Def. 8}}{=} \langle \nabla_{c_\kappa}\|\mathcal{Y}\|_F, \nabla_{G_\kappa}\|\mathcal{Y}\|_F\rangle_Z \\ &\overset{(10)}{=} \left\langle \mathbf{0}, \frac{G_\kappa}{n_\kappa\|\mathcal{Y}\|_F}\right\rangle_Z = \frac{1}{n_\kappa\|\mathcal{Y}\|_F}\langle \mathbf{0}, G_\kappa\rangle_Z. \; \square\end{aligned}$$

The negative gradients of the F-radius of a zonotope point towards the center of the zonotope (Fig. 6); hence, minimizing the F-radius of a zonotope reduces the size of the zonotope. With Prop. 11, we can compute the gradient of a set-based loss function:

Proposition 12 (Set-Based Loss Gradient)**.** *The gradient of the set-based loss function $\widetilde{E}$ is*

$$\nabla_{\mathcal{Y}}\widetilde{E}(t,\mathcal{Y}) = \left\langle (1-\tau)\,\nabla_{c_\kappa}E(t,c_\kappa), \frac{\tau}{\epsilon\, n_\kappa\|\mathcal{Y}\|_F}\,G_\kappa\right\rangle_Z,$$

where $\mathcal{Y} = \langle c_\kappa, G_\kappa\rangle_Z$.

Proof. This follows from Def. 7 and Prop. 11. See the appendix for the details. □

Analogous to the point-based backpropagation Prop. 1, the set-based backpropagation computes for every layer of the neural network the gradient of the set-based loss function $\widetilde{E}$ w.r.t. the output set $\mathcal{H}_k = \langle c_k, G_k\rangle_Z$:

$$\mathcal{G}_k = \langle c_k', G_k'\rangle_Z := \nabla_{\mathcal{H}_k}\widetilde{E}(t,\mathcal{Y}). \qquad (11)$$

The set-based backpropagation of linear layers is straightforward as it just applies a linear map (Prop. 1). However, the backpropagation of nonlinear layers is more involved: while the image enclosure only uses linear approximations, these

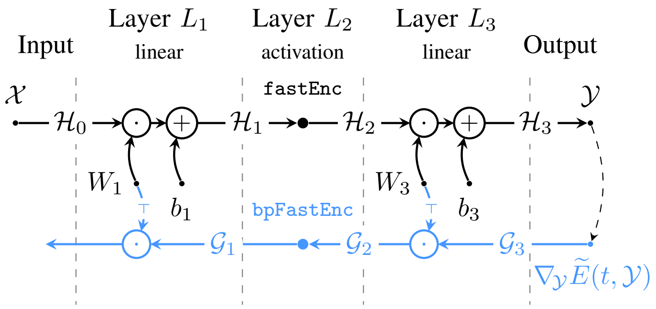

Fig. 7. Visualization of a set-based forward propagation (black) and a set-based backpropagation (blue).

depend on the input set. Thus, intuitively, we have to apply the product rule.

Proposition 13 (Backpropagation through Image Enclosure)**.** *Assume the k-th layer is a nonlinear layer with activation function μ_k. Given an input set $\mathcal{H}_{k-1} = \langle c_{k-1}, G_{k-1} \rangle_Z$ with $G_{k-1} \in \mathbb{R}^{n_{k-1} \times p}$ and a gradient set $\mathcal{G}_k = \langle c'_k, G'_k \rangle_Z$, the gradient set $\mathcal{G}_{k-1} = \langle c'_{k-1}, G'_{k-1} \rangle_Z$ is computed for each dimension $i \in [n_k]$ as*

$$\begin{aligned} \mathcal{G}_{k-1(i)} = {} & \left(c'_{k(i)}\, c_{k-1(i)} + G'_{k(i,[p])}\, G^\top_{k-1(i,\cdot)} \right) \nabla_{\mathcal{H}_{k-1(i)}} a_{k(i)} \\ & + \mathcal{G}_{k(i)}\, a_{k(i)} + c'_{k(i)}\, \nabla_{\mathcal{H}_{k-1(i)}} d_{c,k(i)} \\ & + G'_{k(i,p+i)}\, \nabla_{\mathcal{H}_{k-1(i)}} d_{k(i)}, \end{aligned}$$

Proof. See appendix. □

Alg. 2 uses Prop. 13 to compute the backpropagation of our image enclosure. We note that the set-based backpropagation computes the gradients w.r.t. the computations of the set-based forward propagation, which is done exactly, and no new approximation errors are introduced during the set-based backpropagation.

Using the backpropagation of an image enclosure, we can (analogous to Prop. 1) backpropagate the gradient sets through all layers of a neural network. Fig. 7 visualizes the computations during a set-based forward and set-based backpropagation.

Proposition 14 (Set-Based Backpropagation)**.** *Let $\mathcal{Y} \subset \mathbb{R}^{n_\kappa}$ 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*

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

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

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

Proof. See appendix. □

C. Set-Based Update of Weights and Biases

We now describe how the set of gradients $\mathcal{G}_k$ and the set of inputs $\mathcal{H}_{k-1}$ are used to update the weights and biases of a linear layer. The chain rule is also used to derive the gradients of the set-based loss w.r.t. the weights and bias of a linear layer.

Algorithm 2: Backpropagation of gradients through the fast image enclosure (Alg. 1) of a nonlinear layer.

```
1 function bpFastEnc($\mu_k, \mathcal{G}_k$)
2     Obtain input set $\mathcal{H}_{k-1}$ and slope $a_k$.            // Alg. 1
3     Compute gradients of slope and approx. errors:
        $\nabla_{\mathcal{H}_{k-1}} a_{k(i)}$, $\nabla_{\mathcal{H}_{k-1}} d_{c,k(i)}$, $\nabla_{\mathcal{H}_{k-1}} d_{k(i)}$.   // Prop. 13
4     Add gradients to compute $\mathcal{G}_{k-1}$.                  // Prop. 13
```

Proposition 15 (Gradients w.r.t. Weights and Bias)**.** *The gradients of the set-based loss w.r.t. a weight matrix and a bias vector are*

$$\nabla_{W_k} \widetilde{E}(t, \mathcal{Y}) = c'_k\, c^\top_{k-1} + G'_k\, G^\top_{k-1}, \quad \nabla_{b_k} \widetilde{E}(t, \mathcal{Y}) = c'_k,$$

where $\mathcal{G}_k = \langle c'_k, G'_k \rangle_Z$ and $\mathcal{H}_{k-1} = \langle c_{k-1}, G_{k-1} \rangle_Z$.

Proof. See appendix. □

The weight matrices and bias vectors are updated analogous to point-based training (3) using the gradients of the set-based loss function:

$$\begin{aligned} W_k & \leftarrow W_k - \eta\, \nabla_{W_k} \widetilde{E}(t, \mathcal{Y}) \\ & \overset{\text{Prop. 15}}{=} W_k - \eta \left(c'_k\, c^\top_{k-1} + G'_k\, G^\top_{k-1} \right), \\ b_k & \leftarrow b_k - \eta\, \nabla_{b_k} \widetilde{E}(t, \mathcal{Y}) \overset{\text{Prop. 15}}{=} b_k - \eta\, c'_k. \end{aligned} \tag{12}$$

D. Computational Complexity

Alg. 3 implements an iteration of set-based training. First, a set-based forward propagation computes the output set $\mathcal{Y}$ for an ϵ-perturbance set (Lines 1–6). With $\mathcal{Y}$, 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 14–18). To derive the time complexity of Alg. 3, we first derive the time complexity of Alg. 2.

Proposition 16 (Time Complexity of Alg. 2)**.** *Alg. 2 has time complexity $\mathcal{O}(n_{k-1}\, n_k\, q)$, w.r.t. the number of input neurons n_k and the number of generators q.*

Proof. The input set and the slope of the linear approximation can be stored during the forward propagation. The the gradients are computed using only element-wise operations and take constant time for each entry of the center and the generator matrices; there are $n_k + n_k\, q$ entries for each gradient: $3\, \mathcal{O}(n_k + n_k\, q)$ (Line 3). The gradient of the slope is multiplied with the product of the centers and generator matrices (Prop. 13): the computation of the product takes time $\mathcal{O}(n_k\, q)$ for each dimension; hence, in total $n_k\, \mathcal{O}(n_k\, q) = \mathcal{O}(n_k^2\, q)$. Thus, in total Alg. 2 takes time $3\, \mathcal{O}(n_k + n_k\, q) + \mathcal{O}(n_k^2\, q) = \mathcal{O}(n_k^2\, q) = \mathcal{O}(n_{k-1}\, n_k\, q)$, since $n_{k-1} = n_k$ for nonlinear layers. □

Proposition 17 (Time Complexity of Alg. 3)**.** *The zonotopes used in Alg. 3 have at most $q \leq n_0 + \sum_{k \in [\kappa]} n_k$ number of generators. Let $n_{max} := \max_{k \in [\kappa]} n_{k-1}\, n_k$ be the maximum*

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  $\mathcal{H}_0 \leftarrow \langle x, \epsilon\, I_{n_0} \rangle_Z$                        // (5)
2  for $k \leftarrow 1$ to $\kappa$ do              // set-based forward prop. (Prop. 6)
3      if k-th layer is linear then
4          $\mathcal{H}_k \leftarrow W_k\, \mathcal{H}_{k-1} + b_k$
5      else
6          $\mathcal{H}_k \leftarrow \mathtt{fastEnc}(\mu_k, \mathcal{H}_{k-1})$
7  $\mathcal{Y} \leftarrow \mathcal{H}_\kappa$
8  $\mathcal{G}_\kappa \leftarrow \nabla_{\mathcal{Y}} \widetilde{E}(t, \mathcal{Y})$                    // Prop. 12
9  for $k \leftarrow \kappa$ to $1$ do              // set-based backprop. (Prop. 14)
10     if k-th layer is linear then
11         $\mathcal{G}_{k-1} \leftarrow W_k^\top\, \mathcal{G}_k$
12     else
13         $\mathcal{G}_{k-1} \leftarrow \mathtt{bpFastEnc}(\mu_k, \mathcal{G}_k)$
14 for $k \leftarrow 1$ to $\kappa$ do              // update weights and biases (12)
15     if k-th layer is linear then
16         For $\mathcal{G}_k = \langle c'_k, G'_k \rangle_Z$ and $\mathcal{H}_{k-1} = \langle c_{k-1}, G_{k-1} \rangle_Z$
17         $W_k \leftarrow W_k - \eta\, \left(c'_k\, c_{k-1}^\top + G'_k\, G_{k-1}^\top\right)$
18         $b_k \leftarrow b_k - \eta\, c'_k$
```

size of a weight matrix in the neural network. Moreover, Alg. 3 has time complexity $\mathcal{O}(n_{max}\, q\, \kappa)$ w.r.t. n_{max}, q and the number of layers κ.

Proof. The initial ϵ-perturbance set has n_0 generators (Line 1) and every nonlinear layer adds n_k new generators for the approximation errors (Alg. 1). Moreover, there are at most κ nonlinear layers. Thus, in total, there are at most $(n_0 + \sum_{k \in [\kappa]} n_k)$ generators.

Time Complexity: The k-th step of the set-based forward propagation takes time $\mathcal{O}(n_{\text{max}}\, q)$: The linear map (Line 4) as well as the image enclosure (Line 6) takes time $\mathcal{O}(n_{k-1}\, n_k\, q) = \mathcal{O}(n_{\text{max}}\, q)$ (Prop. 4 and 10). Hence, the set-based forward propagation (Lines 2–6) takes time $\kappa\, \mathcal{O}(n_{\text{max}}\, q)$. The gradient of the set-based loss has $(n_\kappa + n_\kappa\, q)$ entries and the computation of each entry takes constant time; hence, computing the gradient takes time $\mathcal{O}(n_\kappa + n_\kappa\, q)$. The k-th step of the set-based backpropagation takes at most $\mathcal{O}(n_{k-1}\, n_k\, q) = \mathcal{O}(n_{\text{max}}\, q)$ time: a linear layer computes a linear map (Line 11), which takes time $\mathcal{O}(n_{k-1}\, n_k\, q)$ (Prop. 4), and the set-based backpropagation of an image enclosure (Line 13) takes time $\mathcal{O}(n_{k-1}\, n_k\, q)$ (Prop. 16). Hence, the set-based backpropagation (Lines 9–13) takes time $\kappa\, \mathcal{O}(n_{\text{max}}\, q)$. Updating a weight matrix takes time $\mathcal{O}(n_{k-1}\, n_k + n_{k-1}\, n_k\, q) = \mathcal{O}(n_{\text{max}}\, q)$ (Line 17) and updating a bias vector takes time $\mathcal{O}(n_k) = \mathcal{O}(n_{\text{max}})$ (Line 18). There are at most κ linear layers; hence, updating the weight matrix and bias vector of all linear layers takes time $\kappa\, (\mathcal{O}(n_{\text{max}}\, q) + \mathcal{O}(n_{\text{max}})) = \kappa\, \mathcal{O}(n_{\text{max}}\, q)$. Thus, in total, an iteration of set-based training takes time $3\, \kappa\, \mathcal{O}(n_{\text{max}}\, q\, \kappa) = \mathcal{O}(n_{\text{max}}\, q\, \kappa)$. □

The time complexity of set-based training is polynomial, and compared to point-based training, only has an additional factor $q \in \mathcal{O}(n_0 + \sum_{k \in [\kappa]} n_k)$. 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 [36].

It is worth noting that set-based training only uses the following 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.

V. Evaluation

We use the MATLAB toolbox CORA [37] to implement set-based training. The efficacy of set-based training is evaluated by training neural networks of three different sizes (Tab. I) on three different datasets: MNIST [38], Street View House Numbers (SVHN) [39], and CIFAR10 [40]. The training parameters can be found in the appendix.

We compare set-based training *Set (our)* against standard point-based training *Point* and four other training approaches: PGD [19], TRADES [26], IBP [21], and SABR [25].

Moreover, we distinguish between the perturbation radius ϵ_{train} used during training and the perturbation radius ϵ_{test} used during testing. All reported perturbation radii are w.r.t. normalized inputs between 0 and 1. Ideally, we would like to report the adversarial accuracy of a neural network 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 [24]. Therefore, we report (i) a verified accuracy as a lower bound and (ii) a falsified accuracy as an upper bound for the adversarial accuracy. The (i) verified accuracy is the percentage of test inputs for which we can formally verify the robustness of the neural network (in polynomial time). For IBP [21] and SABR [25], we use interval bound propagation [21, Sec. 3] to compute the verified accuracies[1], while we use set-based computations with zonotopes [16, Prop. B.2] for all other approaches (Point, PGD [19], TRADES [26], and Set (our)). Furthermore, the (ii) falsified accuracy is the accuracy of a neural network for adversarial attacks computed with PGD [18]. Fig. 9 illustrates the relationship between verified and falsified accuracy.

Tab. II to IV report our results. We list the findings of our experiments:

- Across all datasets and network sizes, PGD and TRADES consistently achieve the highest clean ($\epsilon_{\text{test}} = 0.0$) and falsified accuracies, e.g., nn-med with MNIST (Tab. II) or nn-med with SVHN (Tab. III). However, their verified accuracies are among the lowest, i.e., 0 for nn-med with MNIST (Tab. II). This behavior is illustrated in Fig. 8. In most cases, set-based training matches the clean

[1] We note that our results show lower verified accuracies for IBP and SABR compared to their reported results. However, this is because [21] uses mixed-integer programming and [25] uses a branch-and-bound algorithm to verify their networks; both these verifiers have worst-case exponential runtime.

TABLE I
NETWORKS AND THEIR NUMBER OF PARAMETERS.

Dataset	Model	#Hidden Neurons	#Parameters
MNIST	nn-small nn-med nn-large	2 × 100 5 × 100 7 × 250	89 610 119 910 575 260
SVHN / CIFAR-10	nn-med nn-large	5 × 100 7 × 250	348 710 1 147 260

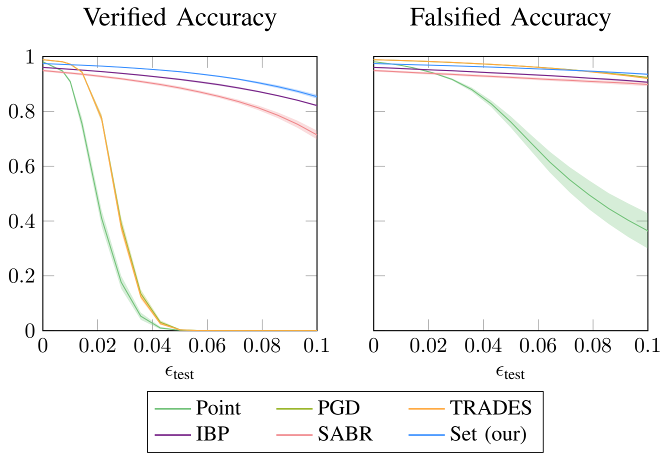

Fig. 8. Results for nn-large trained on MNIST.

accuracies of PGD and TRADES; in some cases, set-based training even achieves higher falsified accuracies, e.g., nn-large with CIFAR10 (Tab. IV).

- Set-based training consistently admits high verified accuracies and achieves higher verified accuracies compared to IBP and SABR, e.g., nn-med with SVHN (Tab. III); with nn-small on MNIST being an exception (Tab. II).
- For MNIST and CIFAR10, we include the results for nn-med reported by [24], which utilize set-based computing with zonotopes. In both cases, set-based training outperforms the results in [24], which confirms the hypothesis that more set-based information is beneficial for the robustness of a neural network.
- Interestingly, the verified accuracies of set-based training with MNIST increase with the size of the neural networks. Usually, larger neural networks are harder to verify because larger approximation errors accumulate. Thus, this indicates that set-based trained neural networks are easier to verify with polynomial-time algorithms.

a) Training Times: A comparison of the training times shows that set-based training is the slowest (Tab. V). However, this is expected because set-based training scales quadratically in the number of input dimensions due to the number of generators used during training (Prop. 17).

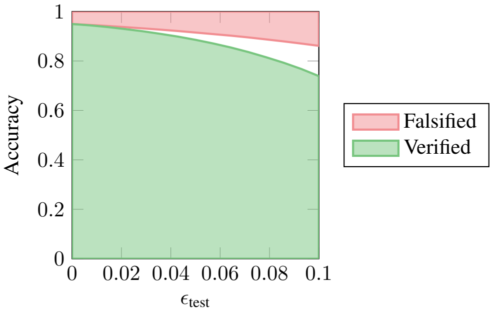

Fig. 9. Comparing the verified and falsified accuracy (nn-small trained set-based on MNIST).

TABLE II
VERIFIED AND FALSIFIED ACCURACIES FOR MNIST [%].

		$\epsilon_{\text{test}} = 0.0$	$\epsilon_{\text{test}} = 0.1$	
ϵ_{train}	**Method**		verified	falsified
nn-small				
0.0	Point	97.81±0.08	0.00±0.00	5.70±0.78
0.1	PGD [19]	**98.58**±0.04	0.61±0.21	**92.17**±0.16
	TRADES [26]	**98.57**±0.08	0.41±0.15	91.86±0.12
	IBP [21]	96.25±0.19	**84.03**±0.43	90.88±0.19
	SABR [25]	96.09±0.19	80.98±0.53	91.52±0.33
	Set (our)	96.04±0.13	76.10±0.99	88.75±0.38
nn-med				
0.0	Point	97.60±0.12	0.00±0.00	21.12±5.51
0.1	PGD [19]	**98.57**±0.07	0.00±0.00	**92.36**±0.17
	TRADES [26]	**98.56**±0.10	0.00±0.00	**92.03**±0.16
	[24][2]	94.40	67.00	86.40
	IBP [21]	95.53±0.25	**81.76**±1.20	89.69±0.50
	SABR [25]	94.65±0.44	75.16±1.84	89.58±0.62
	Set (our)	96.02±0.19	**81.04**±1.29	90.67±0.65
nn-large				
0.0	Point	98.09±0.11	0.00±0.00	36.48±6.34
0.1	PGD [19]	**98.83**±0.04	0.00±0.00	92.35±0.16
	TRADES [26]	**98.85**±0.07	0.00±0.00	92.03±0.19
	IBP [21]	96.03±0.24	82.11±0.32	90.57±0.32
	SABR [25]	94.89±0.42	71.39±1.60	89.81±0.67
	Set (our)	97.44±0.09	**85.36**±0.74	**93.54**±0.24

TABLE III
VERIFIED AND FALSIFIED ACCURACIES FOR SVHN [%].

		$\epsilon_{\text{test}} = 0.0$	$\epsilon_{\text{test}} = 0.01$	
ϵ_{train}	**Method**		verified	falsified
nn-med				
0.0	Point	80.78±0.18	0.03±0.01	50.43±0.38
0.01	PGD [19]	**84.66**±0.17	2.16±0.10	**70.37**±0.25
	TRADES [26]	**84.57**±0.17	2.45±0.08	**70.00**±0.20
	IBP [21]	74.35±0.41	52.25±0.37	64.52±0.37
	SABR [25]	74.75±0.25	47.14±0.54	65.23±0.27
	Set (our)	81.88±0.23	**54.95**±0.22	**70.27**±0.23
nn-large				
0.0	Point	81.93±0.15	0.00±0.00	56.98±0.30
0.01	PGD [19]	**84.32**±0.18	0.01±0.00	67.93±0.26
	TRADES [26]	**84.57**±0.12	0.05±0.01	68.50±0.25
	IBP [21]	75.17±0.21	53.28±0.25	66.44±0.27
	SABR [25]	75.30±0.43	47.55±0.46	67.03±0.32
	Set (our)	83.92±0.24	**55.88**±0.23	**72.73**±0.24

[2] Reported in literature; accuracies for the same network architecture, but different training hyperparameters.

[3] As stated in [21]: $\epsilon = {}^{2}/_{255}$ is roughly equivalent to $\epsilon = 0.03$ by [24], where the perturbation is applied after normalization.

TABLE IV
VERIFIED AND FALSIFIED ACCURACIES FOR CIFAR10 [%].

ϵ_{train}	**Method**	$\epsilon_{\text{test}} = 0.0$	$\epsilon_{\text{test}} = 2/255$ verified	$\epsilon_{\text{test}} = 2/255$ falsified
nn-med				
0.0	Point	51.72±0.33	0.21±0.05	18.67±0.40
2/255	PGD [19]	**54.17**±0.39	15.58±0.58	**42.65**±0.32
	TRADES [26]	**53.82**±0.40	10.11±0.55	39.58±0.25
	[24][23]	47.80	10.00	31.2
	IBP [21]	42.80±0.89	27.04±1.35	38.09±0.76
	SABR [25]	45.92±0.46	13.20±0.78	40.48±0.37
	Set (our)	53.13±0.19	**30.69**±0.25	41.17±0.26
nn-large				
0.0	Point	51.00±0.46	0.00±0.00	15.15±0.40
2/255	PGD [19]	**56.23**±0.56	0.03±0.02	39.27±0.29
	TRADES [26]	**56.12**±0.38	0.02±0.01	38.50±0.29
	IBP [21]	29.28±0.57	20.43±1.02	27.73±0.42
	SABR [25]	38.67±2.19	10.99±1.76	34.50±1.86
	Set (our)	55.58±0.29	**26.22**±0.26	**41.71**±0.23

TABLE V
COMPARING THE TRAINING TIME WITH NN-MED ON MNIST AND NN-LARGE ON CIFAR10 (AVERAGE OF 10 TRAINING RUNS) [SEC / EPOCH].

	Training Time [sec / Epoch]	
Method	nn-large, MNIST	nn-large, CIFAR10
Point	2.8	3.2
PGD [19]	13.8	18.9
TRADES [26]	13.6	20.1
IBP [21]	8.0	11.2
SABR [25]	16.1	25.4
Set (our)	20.0	421.1

VI. CONCLUSION

This paper introduces the first training procedure for robust neural networks using sets for the entire training process. We use set-based computations with zonotopes to efficiently compute output sets for which we compute a set-based loss which extends the well-known tradeoff-loss [26]. Sets of gradients are then backpropagated through the neural network for training. The efficient propagation of sets through a neural network is made possible by an image enclosure that can be evaluated efficiently for entire batches of input sets. Moreover, we prove that the approximation errors of our image enclosure are always smaller or equal compared to Singh's enclosure [14, Thm. 3.2]. Our experimental results demonstrate that our set-based approach effectively trains robust neural networks, which can be easily verified using simple polynomial-time verification algorithms. Moreover, neural networks trained with our set-based training consistently match or outperform those trained with state-of-the-art robust training approaches; thereby, we demonstrate that sets can be effectively used to train robust neural networks. Hence, set-based training represents a promising new direction for the field of robust neural network training.

ACKNOWLEDGEMENTS

This work was partially supported by the project SPP-2422 (No. 500936349) and the project FAI (No. 286525601), both funded by the German Research Foundation (DFG). We also want to thank our colleagues Mark Wetzlinger from our research group for their revisions of the manuscript.

REFERENCES

[1] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[2] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "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)*, 2023, pp. 7464–7475.

[3] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2014.

[4] C. Ye, Y. Wang, Y. Wang, and M. Tie, "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*, vol. 236, no. 9, pp. 1991–2011, 2022.

[5] A. Irfan, K. D. Julian, H. Wu, C. Barrett, M. J. Kochenderfer, B. Meng, and J. Lopez, "Towards verification of neural networks for small unmanned aircraft collision avoidance," in *AIAA/IEEE Digital Avionics Systems Conference (DASC)*, 2020, pp. 1–10.

[6] C. Brix, M. N. Müller, S. Bak, T. T. Johnson, and C. Liu, "First three years of the international verification of neural networks competition (VNN-COMP)," *Int. Journal on Software Tools for Technology Transfer*, vol. 25, no. 3, pp. 329–339, 2023.

[7] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *Int. Conf. on Computer Aided Verification (CAV)*, 2017, pp. 97–117.

[8] G. Singh, T. Gehr, M. Püschel, and M. Vechev, "Boosting robustness certification of neural networks," in *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2019.

[9] H. Zhang, S. Wang, K. Xu, L. Li, B. Li, S. Jana, C.-J. Hsieh, and J. Z. Kolter, "General cutting planes for bound-propagation-based neural network verification," *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1656–1670, 2022.

[10] R. Bunel, I. Turkaslan, P. H. S. Torr, M. P. Kumar, J. Lu, and P. Kohli, "Branch and bound for piecewise linear neural network verification," *Journal of Machine Learning Research*, vol. 21, no. 1, 2020.

[11] C. Ferrari, M. N. Müller, N. Jovanović, and M. Vechev, "Complete verification via multi-neuron relaxation guided branch-and-bound," in *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2022.

[12] A. Girard, "Reachability of uncertain linear systems using zonotopes," in *Proc. of the Int. Conf. on Hybrid Systems: Computation and Control (HSCC)*, 2005, pp. 291–305.

[13] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," in *IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 3–18.

[14] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[15] N. Kochdumper, C. Schilling, M. Althoff, and S. Bak, "Open- and closed-loop neural network verification using polynomial zonotopes," in *NASA Formal Methods*, 2023, pp. 16–36.

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

[17] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2015.

[18] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," in *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2017.

[19] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2018.

[20] L. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, L. Daniel, D. Boning, and I. Dhillon, "Towards fast computation of certified robustness for ReLU networks," in *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2018, pp. 5276–5285.

[21] S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelovic, T. A. Mann, and P. Kohli, "Scalable verified training for provably robust image classification," in *Proc. of the IEEE/CVF Int. Conf. on Computer Vision (ICCV)*, 2019, pp. 4841–4850.

[22] H. Zhang, H. Chen, C. Xiao, S. Gowal, R. Stanforth, B. Li, D. S. Boning, and C. Hsieh, "Towards stable and efficient training of verifiably robust neural networks," in *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2020.

[23] E. Wong and Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2018, pp. 5286–5295.

[24] M. Mirman, T. Gehr, and M. Vechev, "Differentiable abstract interpretation for provably robust neural networks," in *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2018, pp. 3578–3586.

[25] M. N. Müller, F. Eckert, M. Fischer, and M. Vechev, "Certified training: Small boxes are all you need," in *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2023.

[26] H. Zhang, Y. Yu, J. Jiao, E. Xing, L. E. Ghaoui, and M. Jordan, "Theoretically principled trade-off between robustness and accuracy," in *Proc. of the Int. Conf. on Machine Learning (ICML)*, vol. 97, 2019, pp. 7472–7482.

[27] Z. Shi, Y. Wang, H. Zhang, J. Yi, and C.-J. Hsieh, "Fast certified robust training with short warmup," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021, pp. 18 335–18 349.

[28] A. Ross and F. Doshi-Velez, "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)*, 2018, pp. 1660–1669.

[29] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 582–597.

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

[31] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. of the Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*. PMLR, 2010, pp. 249–256.

[32] M. Althoff, "Reachability analysis and its application to the safety assessment of autonomous cars," Ph.D. dissertation, Technische Universität München, 2010.

[33] G. Elekes, "A geometric inequality and the complexity of computing volume," *Discrete & Computational Geometry*, vol. 1, no. 4, pp. 289–292, 1986.

[34] C. Combastel, "Zonotopes and Kalman observers: Gain optimality under distinct uncertainty paradigms and robust convergence," *Automatica*, vol. 55, pp. 265–273, 2015.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016.

[36] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018, pp. 4944–4953.

[37] M. Althoff, "An introduction to CORA 2015," in *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems (ARCH)*, 2015, pp. 120–151.

[38] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[39] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. of the NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[40] A. Krizhevsky, "Learning multiple layers of features from tiny images," Computer Science Department, University of Toronto, Tech. Rep., 2009.

[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2015.

[42] C. Müller, F. Serre, G. Singh, M. Püschel, and M. Vechev, "Scaling polyhedral neural network verification on gpus," in *Proc. Machine Learning and Systems (MLSys)*, 2021, pp. 733–746.

Appendix A
Evaluation Details

a) Hardware: Our experiments were run on a server with 2×AMD EPYC 7763 (64 cores/128 threads), 2TB RAM, and a NVIDIA A100 40GB GPU.

b) Training Hyperparameters: We used the same hyperparameters across all training methods and datasets: we train 100 epochs with a mini-batch size 128. The weights and biases are initialized as in [27]. We use Adam optimizer [41] with the recommended hyperparameters. The initial learning rate of $\eta = 10^{-3}$ is decayed twice by 0.1 at epochs 60 and 80, except for CIFAR10 where we use an initial learning rate $\eta = 10^{-4}$. The first 5 epochs are trained without any perturbation (*warm-up*). The training perturbation radius is linearly increased from epoch 6 to epoch 20 (*ramp-up*). Our hyperparameter τ is set to 0.5 for MNIST and $\tau = 0.1$ for CIFAR-10 and SVHN. For all training methods, we tried to use hyperparameters as close as possible to the reported hyperparameters in their respective paper: for [21] we used $\kappa = 1/2$; for [26] we use $1/\lambda = 6$; for [25] we use $\lambda = 0.1$ for CIFAR10 and $\lambda = 0.4$ for MNIST and SVHN. We did not use any weight decay, and we omitted the *elision of the last layer* for [21]. For any PGD during training (used by methods [19], [26], [25]) we used the settings from [25]: 8 iterations with an initial step size 0.5, which is decayed twice by 0.1 at iterations 4 and 7. All PGD attacks for testing are computed with 40 iterations of step size 0.01. Moreover, all reported accuracies are averaged over 10 runs. Each linear layer is followed by a nonlinear layer except for the last one. For all networks, we use the ReLU activation function.

c) Datasets: MNIST contains $60\,000$ grayscale images of size 28×28. Each image depicts a handwritten digit from 0 to 9. SVHN is a real-world dataset that contains $73\,257$ colored images of digits of house numbers that are cropped to size 32×32. The CIFAR10 dataset contains $60\,000$ colored images of size 32×32. We use the canonical split of training and test data for each dataset and the entire test data for evaluation. After applying the perturbation, we normalize the inputs of SVHN and CIFAR10 with the mean ($\mu = 0.5$ and $\mu = 0.4734$) and standard deviation ($\sigma = 0.2$ and $\sigma = 0.2516$): $\frac{x_i - \mu}{\sigma}$ for and input x_i. The perturbation is applied before the normalization to ensure comparability with the literature.

d) Limitations: The comparability of our evaluation results with other works is limited. Most other works use large convolutional neural networks (CNN), whereas we only evaluate set-based training for feed-forward neural networks with linear layers and nonlinear activation layers. Moreover, some other works use special input data normalization, e.g., [24], which changes the perturbation radius, preventing a meaningful comparison of accuracies. Furthermore, we do not use any data augmentation for SVHN and CIFAR10 like [21], [25].

e) Fairness: For the evaluation, we use our own implementation of the approaches [19], [21], [26], [25]. The implementations are validated by reproducing their reported results. For all approaches, we used the reported training hyperparameters without further tuning. Moreover, we note that our set-based training approach has the fewest extra hyperparameters, i.e., compared to standard point-based training, set-based training only uses two extra hyperparameters: perturbation radius ϵ and τ for weighting the set-based loss function. The number of warm-up and ramp-up epochs are shared among all training approaches. We note that the reported verified accuracies in [21], [25] are computed using a mixed-integer programming (MIP) or branch-and-bound (BnB) verification approach, which can take up to 34h for MNIST networks [42, Appendix C: Hardware and Timing]; we only report verified accuracies computed with interval bound propagation.

Appendix B
Reproducibility of Fig. 5

Fig. 5 compares the decision boundaries of a point-based and a set-based trained neural network for a binary classification task. The network architecture is nn-med: 5 layers with 100 neurons each. The training data are 20 random input samples $x_i \in [0,1]^2$ with corresponding targets $t_i \in \{0,1\}^2$:

$$\begin{aligned}\mathcal{D} = \Big\{ &\left(\begin{bmatrix}0.0622\\0.6995\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}\right), \left(\begin{bmatrix}0.6534\\0.9409\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}\right),\\ &\left(\begin{bmatrix}0.4759\\0.7163\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}\right), \left(\begin{bmatrix}0.8812\\0.1020\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}\right),\\ &\left(\begin{bmatrix}0.5047\\0.4685\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}\right), \left(\begin{bmatrix}0.1470\\0.3275\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}\right),\\ &\left(\begin{bmatrix}0.3439\\0.1395\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}\right), \left(\begin{bmatrix}0.9098\\0.5422\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}\right),\\ &\left(\begin{bmatrix}0.8588\\0.8696\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}\right), \left(\begin{bmatrix}0.0545\\0.0825\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}\right),\\ &\left(\begin{bmatrix}0.6889\\0.4771\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}\right), \left(\begin{bmatrix}0.9329\\0.2857\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}\right),\\ &\left(\begin{bmatrix}0.6781\\0.3043\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}\right), \left(\begin{bmatrix}0.4641\\0.3302\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}\right),\\ &\left(\begin{bmatrix}0.4575\\0.9487\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}\right), \left(\begin{bmatrix}0.1272\\0.4699\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}\right),\\ &\left(\begin{bmatrix}0.6506\\0.7315\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}\right), \left(\begin{bmatrix}0.5207\\0.1229\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}\right),\\ &\left(\begin{bmatrix}0.3271\\0.4574\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}\right), \left(\begin{bmatrix}0.6858\\0.0616\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}\right)\Big\}.\end{aligned}$$

We train both neural networks for 200 epochs with a mini-batch size of 10 using Adam optimizer with a learning rate of $\eta = 0.01$. For set-based training, we use $\epsilon = 0.05$ with $\tau = 0.1$.

Appendix C
Reproducibility of Fig. 4

Fig. 4 compares the output sets computed with our image enclosure and Singh's enclosure. We use the point-based trained neural network from our first training run of nn-med on MNIST. The depicted output sets are computed for the 8-th image of the test set with a perturbation radius $\epsilon = 0.005$.

Appendix D
Proofs of Sec. III

Theorem 1. *Let μ be an s-shaped function, and let $[l, u]$ be an interval. Moreover, let $\underline{d}$ and $\overline{d}$ be the approximation errors of p as defined in Def. 5 and 6, and let d_S be the approximation error of Singh's enclosure [14, Thm. 3.2]. It holds that*

$$\operatorname{area}([\underline{d}, \overline{d}], [l, u]) \leq \operatorname{area}([-d_S, d_S], [l, u]).$$

Proof. We first observe that the approximation errors $\underline{d}$ and $\overline{d}$ of p can be computed at points $\overline{x}, \underline{x} \in [l, u]$ such that $\underline{x} \leq \overline{x}$ (see Fig. 10):

$$\underline{d} = \mu(\underline{x}) - p(\underline{x}), \qquad \overline{d} = \mu(\overline{x}) - p(\overline{x}). \tag{13}$$

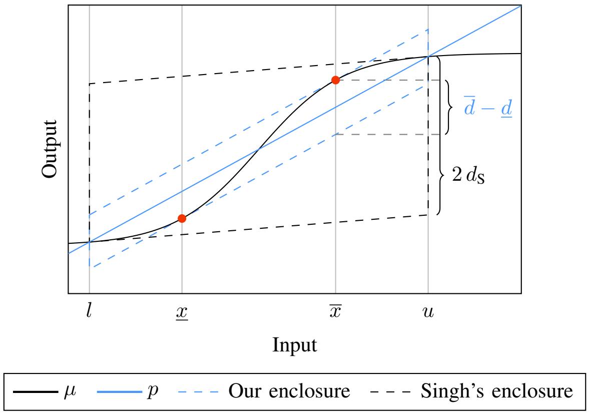

Fig. 10. Illustration for Theorem 1.

Singh's enclosure [14, Thm. 3.2] uses the linear approximation $p_S(x) = a_S\,x + b_S$, with slope a_S, offset b_S, and approximation error d_S:

$$\begin{aligned} a_S &= \min(\mu'(l), \mu'(u)) \\ b_S &= 1/2\,(\mu(u) + \mu(l) - a_S\,(u + l)) \\ d_S &= 1/2\,(\mu(u) - \mu(l) - a_S\,(u - l)). \end{aligned} \tag{14}$$

With Def. 6, we have the following inequality:

$$a = \frac{\mu(u) - \mu(l)}{u - l} \geq a_S.$$

Hence, we have

$$a\,(\overline{x} - \underline{x}) \geq a_S\,(\overline{x} - \underline{x}). \tag{15}$$

Moreover, from (14) we have for all $x \in [l, u]$:

$$\begin{aligned} &\mu(x) - \mu(l) \geq a_S\,(x - l) \\ &\qquad \implies \mu(\underline{x}) \geq \mu(l) + a_S\,(\underline{x} - l), \\ &\mu(u) - \mu(x) \geq a_S\,(u - x) \\ &\qquad \implies \mu(\overline{x}) \leq \mu(u) - a_S\,(u - \overline{x}). \end{aligned} \tag{16}$$

Ultimately, we have

$$\begin{aligned} \overline{d} - \underline{d} &\overset{(13)}{=} \mu(\overline{x}) - p(\overline{x}) - (\mu(\underline{x}) - p(\underline{x})) \\ &\overset{\text{Def. 6}}{=} \mu(\overline{x}) - (a\,\overline{x}) - (\mu(\underline{x}) - (a\,\underline{x})) \\ &= \mu(\overline{x}) - \mu(\underline{x}) - a\,(\overline{x} - \underline{x}) \\ &\overset{(15)}{\leq} \mu(\overline{x}) - \mu(\underline{x}) - a_S\,(\overline{x} - \underline{x}) \\ &\overset{(16)}{\leq} \mu(u) - a_S\,(u - \overline{x}) - (\mu(l) + a_S\,(\underline{x} - l)) \\ &\qquad\qquad - a_S\,(\overline{x} - \underline{x}) \\ &= \mu(u) - \mu(l) - a_S\,(u - l) \\ &\overset{(14)}{=} 2\,d_S. \end{aligned} \tag{17}$$

Hence, we obtain the bound $\overline{d} - \underline{d} \leq 2\,d_S$. Thus,

$$\begin{aligned} \operatorname{area}([\underline{d}, \overline{d}], [l, u]) &\overset{(8)}{=} (u - l)\,(\overline{d} - \underline{d}) \\ &\overset{(17)}{\leq} (u - l)\,2\,d_S \\ &\overset{(8)}{=} \operatorname{area}([-d_S, d_S], [l, u]). \end{aligned} \tag{18}$$

□

Appendix E
Proofs of Sec. IV

We first prove the correctness of the gradient of the set-based loss.

Proposition 12 (Set-Based Loss Gradient)**.** *The gradient of the set-based loss function $\widetilde{E}$ is*

$$\nabla_{\mathcal{Y}} \widetilde{E}(t, \mathcal{Y}) = \left\langle (1-\tau)\, \nabla_{c_\kappa} E(t, c_\kappa), \frac{\tau}{\epsilon\, n_\kappa\, \|\mathcal{Y}\|_F}\, G_\kappa \right\rangle_Z,$$

where $\mathcal{Y} = \langle c_\kappa, G_\kappa \rangle_Z$.

Proof. This follows from Def. 7 and Prop. 11:

$$\begin{aligned}
\nabla_{\mathcal{Y}} \widetilde{E}(t, \mathcal{Y}) &\overset{\text{Def. 7}}{=} (1-\tau)\, \nabla_{\mathcal{Y}} E(t, c_\kappa) + \frac{\tau}{\epsilon}\, \nabla_{\mathcal{Y}} \|\mathcal{Y}\|_F \\
&\overset{\text{Def. 8}}{=} (1-\tau)\, \langle \nabla_{c_\kappa} E(t, c_\kappa), \mathbf{0} \rangle_Z \\
&\quad + \frac{\tau}{\epsilon}\, \nabla_{\mathcal{Y}} \|\mathcal{Y}\|_F \\
&\overset{\text{Prop. 11}}{=} (1-\tau)\, \langle \nabla_{c_\kappa} E(t, c_\kappa), \mathbf{0} \rangle_Z \\
&\quad + \frac{\tau}{\epsilon\, n_\kappa\, \|\mathcal{Y}\|_F}\, \langle \mathbf{0}, G_\kappa \rangle_Z \\
&= \left\langle (1-\tau)\, \nabla_{c_\kappa} E(t, c_\kappa), \frac{\tau}{\epsilon\, n_\kappa\, \|\mathcal{Y}\|_F}\, G_\kappa \right\rangle_Z. \quad \square
\end{aligned}$$

Moreover, in this section, we give three proofs for the set-based backpropagation: (i) set-based backpropagation through an image enclosure (Prop. 13), (ii) set-based backpropagation through all layers of a neural network (Prop. 14), and (iii) set-based weight and bias update (Prop. 15).

Before we prove the propositions required for the set-based backpropagation, we unfold (11) and rewrite the gradient set $\mathcal{G}_{k-1}$ using the chain rule for partial derivatives. Let $\mathcal{H}_k = \langle c_k, G_k \rangle_Z$ with $G_k \in \mathbb{R}^{n_k \times q}$ be the output set of the k-th layer and let $\mathcal{G}_k = \langle c'_k, G'_k \rangle_Z$ be the gradient set w.r.t. $\mathcal{H}_k$:

$$\begin{aligned}
\mathcal{G}_{k-1} &\overset{(11)}{=} \nabla_{\mathcal{H}_{k-1}} \widetilde{E}(t, \mathcal{Y}) \\
&= \sum_{i=1}^{n_k} \frac{\partial \widetilde{E}(t, \mathcal{Y})}{\partial c_{k(i)}}\, \nabla_{\mathcal{H}_{k-1}} c_{k(i)} \\
&\quad + \sum_{i=1}^{n_k} \sum_{j=1}^{q} \frac{\partial \widetilde{E}(t, \mathcal{Y})}{\partial G_{k(i,j)}}\, \nabla_{\mathcal{H}_{k-1}} G_{k(i,j)} \qquad (19) \\
&= \sum_{i=1}^{n_k} c'_{k(i)}\, \nabla_{\mathcal{H}_{k-1}} c_{k(i)} \\
&\quad + \sum_{i=1}^{n_k} \sum_{j=1}^{q} G'_{k(i,j)}\, \nabla_{\mathcal{H}_{k-1}} G_{k(i,j)}.
\end{aligned}$$

Note: the plus symbol $(+)$ between zonotopes denotes their element-wise addition: $\langle c_1, G_1 \rangle_Z + \langle c_2, G_2 \rangle_Z = \langle c_1 + c_2, G_1 + G_2 \rangle_Z$, whereas the Minkowski sum is denoted as $\langle c_1, G_1 \rangle_Z \oplus \langle c_2, G_2 \rangle_Z = \langle c_1 + c_2, [\, G_1\ G_2\,] \rangle_Z$.

Proposition 13 (Backpropagation through Image Enclosure)**.** *Assume the k-th layer is a nonlinear layer with activation function μ_k. Given an input set $\mathcal{H}_{k-1} = \langle c_{k-1}, G_{k-1} \rangle_Z$ with $G_{k-1} \in \mathbb{R}^{n_{k-1} \times p}$ and a gradient set $\mathcal{G}_k = \langle c'_k, G'_k \rangle_Z$, the gradient set $\mathcal{G}_{k-1} = \langle c'_{k-1}, G'_{k-1} \rangle_Z$ is computed for each dimension $i \in [n_k]$ as*

$$\begin{aligned}
\mathcal{G}_{k-1(i)} = &\left(c'_{k(i)}\, c_{k-1(i)} + G'_{k(i,[p])}\, G^\top_{k-1(i,\cdot)} \right) \nabla_{\mathcal{H}_{k-1(i)}} a_{k(i)} \\
&+ \mathcal{G}_{k(i)}\, a_{k(i)} + c'_{k(i)}\, \nabla_{\mathcal{H}_{k-1(i)}} d_{c,k(i)} \\
&+ G'_{k(i,p+i)}\, \nabla_{\mathcal{H}_{k-1(i)}} d_{k(i)},
\end{aligned}$$

Proof. The image enclosure adds n_k generators, hence the input set $\mathcal{H}_{k-1}$ has n_k generators less than gradient set $\mathcal{G}_k$, i.e. $G_{k-1} \in \mathbb{R}^{n_k \times p}$ and $G'_k \in \mathbb{R}^{n_k \times q}$ with $q = p + n_k$. We split (19) into three summands:

$$\begin{aligned}
\mathcal{G}_{k-1,c} &= \sum_{i=1}^{n_k} c'_{k(i)}\, \nabla_{\mathcal{H}_{k-1}} c_{k(i)}, \\
\mathcal{G}_{k-1,p} &= \sum_{i=1}^{n_k} \sum_{j=1}^{p} G'_{k(i,j)}\, \nabla_{\mathcal{H}_{k-1}} G_{k(i,j)}, \\
\mathcal{G}_{k-1,q} &= \sum_{i=1}^{n_k} \sum_{j=p+1}^{q} G'_{k(i,j)}\, \nabla_{\mathcal{H}_{k-1}} G_{k(i,j)}.
\end{aligned}$$

Hence,

$$\mathcal{G}_{k-1} = \mathcal{G}_{k-1,c} + \mathcal{G}_{k-1,p} + \mathcal{G}_{k-1,q}. \qquad (20)$$

Furthermore, the input set $\mathcal{H}_{k-1}$ is enclosed by the interval $[l_{k-1}, u_{k-1}]$, where $l_{k-1} = c_{k-1} - |G_{k-1}|\, \mathbf{1}$ and $u_{k-1} = c_{k-1} + |G_{k-1}|\, \mathbf{1}$ (Prop. 2). Moreover, let $e_i \in \{0,1\}^{n_k}$ be the i-th standard basis vector.

Firstly, we derive the gradient $\nabla_{\mathcal{H}_{k-1}} c_{k(i)}$ needed for $\mathcal{G}_{k-1,c}$, for which we need the gradients of center $c_{k(i)} = a_{k(i)}\, c_{k-1(i)} + d_{c,k(i)}$ w.r.t. the input set $\mathcal{H}_{k-1}$ for each dimension $i \in [n_k]$. The image enclosure is applied for each dimension individually; therefore, we can consider each dimension separately because for any dimensions $i, j \in [n_{k-1}]$, where $i \neq j$:

$$\nabla_{\mathcal{H}_{k-1(j)}} c_{k(i)} = \langle 0, \mathbf{0} \rangle_Z, \quad \nabla_{\mathcal{H}_{k-1(j)}} G_{k(i,\cdot)} = \langle 0, \mathbf{0} \rangle_Z. \qquad (21)$$

Let $i \in [n_k]$ be a fixed dimension and $j \in [p]$ a fixed index of a generator. We require the gradient of the slope $a_{k(i)}$ and the offset $d_{c,k(i)}$. The gradient of the slope $a_{k(i)}$ is:

$$\begin{aligned}
\frac{\partial a_{k(i)}}{\partial c_{k-1(i)}} &\overset{\text{Def. 6}}{=} \frac{\mu'(u_{k-1(i)}) - \mu'(l_{k-1(i)})}{u_{k-1(i)} - l_{k-1(i)}}, \\
\frac{\partial a_{k(i)}}{\partial G_{k-1(i,j)}} &\overset{\text{Def. 6}}{=} \left(\frac{\mu'(u_{k-1(i)}) + \mu'(l_{k-1(i)}) - 2\, a_{k(i)}}{u_{k-1(i)} - l_{k-1(i)}} \right) \\
&\qquad \cdot \operatorname{sign}(G_{k-1(i,j)}).
\end{aligned}$$

Let $\overline{x}_k$ and $\underline{x}_k$ be the points of the approximation errors $\overline{d}_k$ and $\underline{d}_k$:

$$\begin{aligned}
\overline{x}_k &= \operatorname*{arg\,max}_{x \in \mathcal{P}} \mu_k(x) - p_k(x), \\
\underline{x}_k &= \operatorname*{arg\,min}_{x \in \mathcal{P}} \mu_k(x) - p_k(x).
\end{aligned}$$

To prevent repetitions in this proof, let g denote the center or an arbitrary generator of the input set $\mathcal{H}_{k-1}$:

$$g \in \left\{ c_{k-1}, G_{k-1(\cdot,1)}, G_{k-1(\cdot,2)}, \dots, G_{k-1(\cdot,p)} \right\}.$$

For $(x, d) \in \{(\overline{x}_k, \overline{d}_k), (\underline{x}_k, \underline{d}_k)\}$, we apply the chain rule and the product-rule to derive the gradients of the approximation error $d_{(i)}$:

$$\begin{aligned}\frac{\partial d_{(i)}}{\partial g_{(i)}} &= \frac{\partial\big(\mu_k(x_{(i)}) - a_{k(i)}\, x_{(i)}\big)}{\partial g_{(i)}} \\ &= \mu_k'(x_{(i)})\,\frac{\partial x_{(i)}}{\partial g_{(i)}} - \left(\frac{\partial a_{k(i)}}{\partial g_{(i)}}\, x_{(i)} + a_{k(i)}\,\frac{\partial x_{(i)}}{\partial g_{(i)}}\right) \\ &= \big(\mu_k'(x_{(i)}) - a_{k(i)}\big)\,\frac{\partial x_{(i)}}{\partial g_{(i)}} - \frac{\partial a_{k(i)}}{\partial g_{(i)}}\, x_{(i)}.\end{aligned}$$

Please recall the offset $d_{c,k} = 1/2\,\big(\overline{d}_k + \underline{d}_k\big)$ and approximation error $d_k = 1/2\,\big(\overline{d}_k - \underline{d}_k\big)$ (Def. 5); hence,

$$\begin{aligned}\frac{\partial d_{c,k(i)}}{\partial g_{(i)}} &= \frac{1}{2}\left(\frac{\partial \overline{d}_{k(i)}}{\partial g_{(i)}} + \frac{\partial \underline{d}_{k(i)}}{\partial g_{(i)}}\right), \\ \frac{\partial d_{k(i)}}{\partial g_{(i)}} &= \frac{1}{2}\left(\frac{\partial \overline{d}_{k(i)}}{\partial g_{(i)}} - \frac{\partial \underline{d}_{k(i)}}{\partial g_{(i)}}\right).\end{aligned}$$

Using the gradient of the slope $a_{k(i)}$ and the gradient of the offset $d_{c,k(i)}$, we derive the gradient of the center $c_{k(i)}$:

$$\begin{aligned}\frac{\partial c_{k(i)}}{\partial c_{k-1(i)}} &= \frac{\partial\big(a_{k(i)}\, c_{k-1(i)} + d_{c,k(i)}\big)}{\partial c_{k-1(i)}} \\ &= a_{k(i)} + \frac{\partial a_{k(i)}}{\partial c_{k-1(i)}}\, c_{k-1(i)} + \frac{\partial d_{c,k(i)}}{\partial c_{k-1(i)}}, \\ \frac{\partial c_{k(i)}}{\partial G_{k-1(i,j)}} &= \frac{\partial\big(a_{k(i)}\, c_{k-1(i)} + d_{c,k(i)}\big)}{\partial G_{k-1(i,j)}} \\ &= \frac{\partial a_{k(i)}}{\partial G_{k-1(i,j)}}\, c_{k-1(i)} + \frac{\partial d_{c,k(i)}}{\partial G_{k-1(i,j)}}.\end{aligned}$$

Hence, we have

$$\begin{aligned}\nabla_{\mathcal{H}_{k-1(i)}} c_{k(i)} &= \left\langle \frac{\partial c_{k(i)}}{\partial c_{k-1(i)}}, \frac{\partial c_{k(i)}}{\partial G_{k-1(i,\cdot)}} \right\rangle_Z \\ &= a_{k(i)} + \nabla_{\mathcal{H}_{k-1(i)}} a_{k(i)}\, c_{k-1(i)} \\ &\quad + \nabla_{\mathcal{H}_{k-1(i)}} d_{c,k(i)}.\end{aligned} \tag{22}$$

Secondly, we derive the gradient $\nabla_{\mathcal{H}_{k-1(i)}} G_{k(i,j)}$ needed for $\mathcal{G}_{k-1,p}$. Let $j' \in [n_k]$ be a different generator index: $j' \neq j$; the gradient of $G_{k(i,j)}$ is

$$\begin{aligned}\frac{\partial G_{k(i,j)}}{\partial c_{k-1(i)}} &= \frac{\partial\big(a_{k(i)}\, G_{k-1(i,j)}\big)}{\partial c_{k-1(i)}} = \frac{\partial a_{k(i)}}{\partial c_{k-1(i)}}\, G_{k-1(i,j)}, \\ \frac{\partial G_{k(i,j)}}{\partial G_{k-1(i,j)}} &= \frac{\partial\big(a_{k(i)}\, G_{k-1(i,j)}\big)}{\partial G_{k-1(i,j)}} \\ &= a_{k(i)} + \frac{\partial a_{k(i)}}{\partial G_{k-1(i,j)}}\, G_{k-1(i,j)}, \\ \frac{\partial G_{k(i,j)}}{\partial G_{k-1(i,j')}} &= \frac{\partial\big(a_{k(i)}\, G_{k-1(i,j)}\big)}{\partial G_{k-1(i,j')}} = \frac{\partial a_{k(i)}}{\partial G_{k-1(i,j')}}\, G_{k-1(i,j)}.\end{aligned}$$

Hence, we have

$$\begin{aligned}\nabla_{\mathcal{H}_{k-1(i)}} G_{k(i,j)} &= \left\langle \frac{\partial G_{k(i,j)}}{\partial c_{k-1(i)}}, \frac{\partial G_{k(i,j)}}{\partial G_{k-1(i,\cdot)}} \right\rangle_Z \\ &= \nabla_{\mathcal{H}_{k-1(i)}} a_{k(i)}\, G_{k-1(i,j)} \\ &\quad + a_{k(i)}\,\big\langle \mathbf{0}, e_i\, e_j^\top \big\rangle_Z.\end{aligned} \tag{23}$$

Thirdly, we derive $\mathcal{G}_{k-1,q}$. Please recall, the diagonal entries of $G_{k(\cdot,p+[n_k])}$ contain the approximation errors, while the non-diagonal entries are 0; hence, the gradient of any non-diagonal entry $j' \in [q]\colon j' > p \wedge j' \neq p + i$ is 0: $\nabla_{\mathcal{H}_{k-1(i)}} G_{k(i,j')} = \langle 0, \mathbf{0}\rangle_Z$. Hence, we can simplify $\mathcal{G}_{k-1,q}$:

$$\begin{aligned}\mathcal{G}_{k-1,q(i)} &= \sum_{j=p+1}^{q} G'_{k(i,j)}\; \nabla_{\mathcal{H}_{k-1(i)}} G_{k(i,j)} \\ &= G'_{k(i,p+i)}\; \nabla_{\mathcal{H}_{k-1(i)}} d_{k(i)}.\end{aligned} \tag{24}$$

We add $\mathcal{G}_{k-1,c}$ and $\mathcal{G}_{k-1,p}$ together and reorder the terms:

$$\begin{aligned}&\mathcal{G}_{k-1,c(i)} + \mathcal{G}_{k-1,p(i)} \\ &= c'_{k(i)}\; \nabla_{\mathcal{H}_{k-1(i)}} c_{k(i)} + \sum_{j=1}^{p} G'_{k(i,j)}\; \nabla_{\mathcal{H}_{k-1(i)}} G_{k(i,j)} \\ &\overset{(22)}{=} c'_{k(i)}\,\big(a_{k(i)} + \nabla_{\mathcal{H}_{k-1(i)}} a_{k(i)}\, c_{k-1(i)} + \nabla_{\mathcal{H}_{k-1(i)}} d_{c,k(i)}\big) \\ &\quad + \sum_{j=1}^{p} G'_{k(i,j)}\; \nabla_{\mathcal{H}_{k-1(i)}} G_{k(i,j)} \\ &\overset{(23)}{=} c'_{k(i)}\,\big(a_{k(i)} + \nabla_{\mathcal{H}_{k-1(i)}} a_{k(i)}\, c_{k-1(i)} + \nabla_{\mathcal{H}_{k-1(i)}} d_{c,k(i)}\big) \\ &\quad + \sum_{j=1}^{p} G'_{k(i,j)}\,\Big(\nabla_{\mathcal{H}_{k-1(i)}} a_{k(i)}\, G_{k-1(i,j)} + a_{k(i)}\,\big\langle \mathbf{0}, e_i\, e_j^\top \big\rangle_Z\Big) \\ &= c'_{k(i)}\, a_{k(i)} + c'_{k(i)}\; \nabla_{\mathcal{H}_{k-1(i)}} a_{k(i)}\, c_{k-1(i)} \\ &\quad + c'_{k(i)}\; \nabla_{\mathcal{H}_{k-1(i)}} d_{c,k(i)} + a_{k(i)}\,\Big\langle \mathbf{0}, G'_{k(i,\cdot)} \Big\rangle_Z \\ &\quad + \sum_{j=1}^{p} G'_{k(i,j)}\; \nabla_{\mathcal{H}_{k-1(i)}} a_{k(i)}\, G_{k-1(i,j)} \\ &= \nabla_{\mathcal{H}_{k-1(i)}} a_{k(i)}\,\Big(c_{k-1(i)}\, c'_{k(i)} + G^\top_{k-1(i,\cdot)}\, G'_{k(i,[p])}\Big) \\ &\quad + \mathcal{G}_{k(i)}\, a_{k(i)} + c'_{k(i)}\; \nabla_{\mathcal{H}_{k-1(i)}} d_{c,k(i)}.\end{aligned} \tag{25}$$

Finally, we obtain

$$\begin{aligned}\mathcal{G}_{k-1(i)} &\overset{(20)}{=} \mathcal{G}_{k-1,c(i)} + \mathcal{G}_{k-1,p(i)} + \mathcal{G}_{k-1,q(I)} \\ &\overset{(25)}{=} \nabla_{\mathcal{H}_{k-1(i)}} a_{k(i)}\,\Big(c_{k-1(i)}\, c'_{k(i)} + G'_{k(i,[p])}\, G^\top_{k-1(i,\cdot)}\Big) \\ &\quad + \mathcal{G}_{k(i)}\, a_{k(i)} + c'_{k(i)}\; \nabla_{\mathcal{H}_{k-1(i)}} d_{c,k(i)} + \mathcal{G}_{k-1,q(i)} \\ &\overset{(24)}{=} \nabla_{\mathcal{H}_{k-1(i)}} a_{k(i)}\,\Big(c_{k-1(i)}\, c'_{k(i)} + G'_{k(i,[p])}\, G^\top_{k-1(i,\cdot)}\Big) \\ &\quad + \mathcal{G}_{k(i)}\, a_{k(i)} + c'_{k(i)}\; \nabla_{\mathcal{H}_{k-1(i)}} d_{c,k(i)} \\ &\quad + G'_{k(i,p+i)}\; \nabla_{\mathcal{H}_{k-1(i)}} d_{k(i)}.\end{aligned}$$

□

Proposition 14 (Set-Based Backpropagation)**.** *Let $\mathcal{Y} \subset \mathbb{R}^{n_\kappa}$ 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*

$$\begin{aligned}\mathcal{G}_\kappa &= \nabla_{\mathcal{Y}} \widetilde{E}(t, \mathcal{Y}), \\ \mathcal{G}_{k-1} &= \begin{cases} W_k^\top\, \mathcal{G}_k & \textit{if k-th layer is linear,} \\ \texttt{bpFastEnc}(\mu_k, \mathcal{G}_k) & \textit{otherwise,}\end{cases}\end{aligned}$$

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

Proof. If $k = \kappa$, we compute the gradient of the set-based loss according to Prop. 12. We assume $k < \kappa$. Let $\mathcal{G}_k = \langle c'_k, G'_k \rangle_Z$ and $\mathcal{H}_k = \langle c_k, G_k \rangle_Z$.

We split cases on the type of the k-th layer and simplify the terms.

Case (i). The k-th layer is linear. For dimension $i \in [n_k]$, we have

$$\begin{aligned} \nabla_{\mathcal{H}_{k-1}} c_{k(i)} &= \nabla_{\mathcal{H}_{k-1}} \big(W_{k(i,\cdot)}\, c_{k-1} + b_{k(i)}\big) \\ &= \Big\langle W_{k(i,\cdot)}^\top, \mathbf{0} \Big\rangle_Z, \\ \nabla_{\mathcal{H}_{k-1}} G_{k(i,j)} &= \nabla_{\mathcal{H}_{k-1}} \big(W_{k(i,\cdot)}\, G_{k-1(\cdot,j)}\big) \\ &= \Big\langle \mathbf{0}, W_{k(i,\cdot)}^\top\, e_j^\top \Big\rangle_Z. \end{aligned} \tag{26}$$

Thus,

$$\begin{aligned} \mathcal{G}_{k-1} &\overset{(19)}{=} \sum_{i=1}^{n_k} c'_{k(i)}\ \nabla_{\mathcal{H}_{k-1}} c_{k(i)} \\ &\quad + \sum_{i=1}^{n_k} \sum_{j=1}^{q} G'_{k(i,j)}\ \nabla_{\mathcal{H}_{k-1}} G_{k(i,j)} \\ &\overset{(26)}{=} \sum_{i=1}^{n_k} c'_{k(i)} \Big\langle W_{k(i,\cdot)}^\top, \mathbf{0} \Big\rangle_Z \\ &\quad + \sum_{i=1}^{n_k} \sum_{j=1}^{q} G'_{k(i,j)} \Big\langle \mathbf{0}, W_{k(i,\cdot)}^\top\ e_j^\top \Big\rangle_Z \\ &= \left\langle \sum_{i=1}^{n_k} W_{k(i,\cdot)}^\top\, c'_{k(i)}, \sum_{i=1}^{n_k} W_{k(i,\cdot)}^\top\, G'_{k(i,\cdot)} \right\rangle_Z \\ &= \big\langle W_k^\top\, c'_k, W_k^\top\, G'_k \big\rangle_Z \\ &= W_k^\top\, \mathcal{G}_k. \end{aligned}$$

Case (ii). The k-th layer is a nonlinear layer. Prop. 13 proves the correctness.

□

Proposition 15 (Gradients w.r.t. Weights and Bias)**.** *The gradients of the set-based loss w.r.t. a weight matrix and a bias vector are*

$$\nabla_{W_k} \widetilde{E}(t, \mathcal{Y}) = c'_k\, c_{k-1}^\top + G'_k\, G_{k-1}^\top, \quad \nabla_{b_k} \widetilde{E}(t, \mathcal{Y}) = c'_k,$$

where $\mathcal{G}_k = \langle c'_k, G'_k \rangle_Z$ *and* $\mathcal{H}_{k-1} = \langle c_{k-1}, G_{k-1} \rangle_Z$*.*

Proof. We rewrite the gradient by applying the chain rule for partial derivatives:

$$\begin{aligned} \nabla_{W_k} \widetilde{E}(t, \mathcal{Y}) &= \sum_{i=1}^{n_k} c'_{k(i)}\ \nabla_{W_k} c_{k(i)} \\ &\quad + \sum_{i=1}^{n_k} \sum_{j=1}^{q} G'_{k(i,j)}\ \nabla_{W_k} G_{k(i,j)}, \\ \nabla_{b_k} \widetilde{E}(t, \mathcal{Y}) &= \sum_{i=1}^{n_k} c'_{k(i)}\ \nabla_{b_k} c_{k(i)} \\ &\quad + \sum_{i=1}^{n_k} \sum_{j=1}^{q} G'_{k(i,j)}\ \nabla_{b_k} G_{k(i,j)}, \end{aligned}$$

where $G_k \in \mathbb{R}^{n_k \times q}$. Moreover, we have for dimension $i \in [n_k]$ and generator index $j \in [q]$:

$$\begin{aligned} \nabla_{W_k} c_{k(i)} &= \nabla_{W_k} \big(W_{k(i,\cdot)}\, c_{k-1} + b_{k(i)}\big) = e_i\, c_{k-1}^\top, \\ \nabla_{W_k} G_{k(i,j)} &= \nabla_{W_k} \big(W_{k(i,\cdot)}\, G_{k-1(\cdot,j)}\big) = e_i\, G_{k-1(\cdot,j)}^\top, \\ \nabla_{b_k} c_{k(i)} &= \nabla_{b_k} \big(W_{k(i,\cdot)}\, c_{k-1} + b_{k(i)}\big) = e_i, \\ \nabla_{b_k} G_{k(i,j)} &= \nabla_{b_k} \big(W_{k(i,\cdot)}\, G_{k-1(\cdot,j)}\big) = \mathbf{0}, \end{aligned}$$

where $e_i \in \{0,1\}^{n_k}$ is the i-th standard basis vector. Thus,

$$\nabla_{W_k} \widetilde{E}(t, \mathcal{Y}) = c'_k\, c_{k-1}^\top + G'_k\, G_{k-1}^\top, \quad \nabla_{b_k} \widetilde{E}(t, \mathcal{Y}) = c'_k. \ \square$$