Machine Learning (2021) 110:1175–1197
https://doi.org/10.1007/s10994-021-05973-0

Reachable sets of classifiers and regression models: (non-)robustness analysis and robust training

Anna-Kathrin Kopetzki[1] · **Stephan Günnemann**[1]

Received: 18 September 2020 / Revised: 28 January 2021 / Accepted: 18 March 2021 /
Published online: 28 April 2021

Abstract
Neural networks achieve outstanding accuracy in classification and regression tasks. However, understanding their behavior still remains an open challenge that requires questions to be addressed on the robustness, explainability and reliability of predictions. We answer these questions by computing *reachable sets* of neural networks, i.e. sets of outputs resulting from continuous sets of inputs. We provide two efficient approaches that lead to over- and under-approximations of the reachable set. This principle is highly versatile, as we show. First, we use it to analyze and enhance the robustness properties of both classifiers and regression models. This is in contrast to existing works, which are mainly focused on classification. Specifically, we verify (non-)robustness, propose a robust training procedure, and show that our approach outperforms adversarial attacks as well as state-of-the-art methods of verifying classifiers for non-norm bound perturbations. Second, we provide techniques to distinguish between reliable and non-reliable predictions for unlabeled inputs, to quantify the influence of each feature on a prediction, and compute a feature ranking.

Keywords Robustness · Verification · Reachable set · Neural network

1 Introduction

Neural networks are widely used in classification and regression tasks. However, understanding their behavior remains an open challenge and raises questions concerning the robustness, reliability and explainability of their predictions. We address these issues by studying the principle of reachable sets of neural networks: Given a *set* of inputs, what is the *set* of outputs of the neural network.

Editors: Annalisa Appice, Sergio Escalera, Jose A. Gamez, Heike Trautmann.

✉ Anna-Kathrin Kopetzki
kopetzki@in.tum.de

Stephan Günnemann
guennemann@in.tum.de

[1] Department of Informatics, Technical University of Munich, Munich, Germany

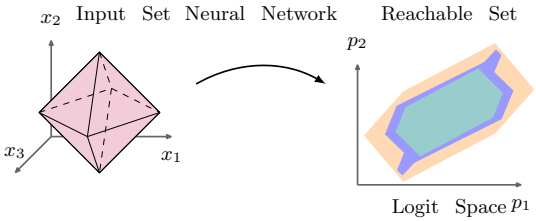


Fig. 1 Orange over-/under-approximation of the reachable set (Color figure online)

Methods that compute an exact reachable set (Xiang et al. 2017) are not feasible, even for tiny neural networks (Liu et al. 2019). In this study, we aim to approximate the reachable set such that it can be computed for neural networks used on standard data sets. More specifically, we investigate this problem in the context of ReLU neural networks, which constitute the most widely used class of networks. To allow flexibility regarding inputs, we propagate a set of points defined by a zonotope through the neural network. As the ReLU operation can result in non-convex sets, we derive under-approximated or over-approximated output sets (see Fig. 1). The resulting sets are used to analyze and enhance neural network properties (see Sect. 4).

Overall, our main contributions are: (i) Two efficient approaches RsO and RsU (Reachable set Over- and Under-approximation) of approximating the reachable set of a neural network; (ii) Classification: Techniques of applying RsU and RsO to (non-)robustness verification, robust training, comparison with attacks, and state-of-the-art verification methods. (iii) Regression: an approach for analyzing and enhancing the robustness properties of regression models. (iv) Explainability/Reliability: a method of distinguishing between reliable and non-reliable predictions as well as a technique of quantifying and ranking features w.r.t. their influence on a prediction.

2 Related work

Reachable sets Computing the exact reachable set of a neural network as Xiang et al. (2017) is not applicable even with tiny networks, as shown in Liu et al. (2019). Some techniques that approximate reachable sets, such as Ruan et al. (2018), cannot handle the common robustness definition. Most approaches that deal with the reachable sets of a neural network emerged from robustness verification. The study that is the most closely related to our over-approximation approach RsO is Gehr et al. (2018). Further developments of this technique include bounds (Singh et al. 2018, 2019a, b, c). In addition, set-based approaches are used for robust training (Gowal et al. 2019; Mirman et al. 2018). Our work goes beyond the existing approaches. First, our over-approximations are (by construction) subsets of the ones computed in Gehr et al. (2018) and thus tighter. Second, in comparison to the improvements presented in Singh et al. (2018, 2019b, c), our approaches do not require bounds on the layer input. Third, in addition to over-approximations, we provide an approach to under-approximate the reachable set.

(Non-)robustness verification Reachable sets are applicable to (non-) robustness verification (see Sect. 3). Other expensive robustness verification methods are based on SMT solvers (Katz et al. 2017; Ehlers 2017; Bunel et al. 2018), mixed integer linear programming (Tjeng et al. 2019) or Lipschitz optimization (Ruan et al. 2018). One family of verification approaches search for adversarial examples by solving the constrained optimization problem of finding a sample that is close to the input, but labeled differently. The search

space, i.e. an over-approximation of the reachable set of the neural network is defined by the constraints. The distance of the samples to an input point is usually bound by a norm that the optimization problem can deal with, such as L_∞-norm (Wong and Kolter 2018; Raghunathan et al. 2018; Bastani et al. 2016; Katz et al. 2017; Steinhardt et al. 2017) or L_2 -norm (Hein and Andriushchenko 2017). One drawback of these approaches is the strong norm-based restriction on the inputs. Our approaches can handle input sets equivalent to norms as well as input sets that couple features and thus allow complex perturbations such as different brightness of pictures.

The complement of robustness verification are adversarial attacks, i.e. points close to an input that are assigned a different label. Adversarial attacks compute a single point within the reachable set, without explicitly computing the reachable set. There are various ways of designing attacks, one of the strongest being the projected gradient descent attack (PGD) (Madry et al. 2018). In contrast to attacks, our RsU approach aims to find an entire set of predictions corresponding to an input set.

It should be noted that, all the above principles are designed for classification tasks. In contrast, our approach is naturally suited for regression as well. To further highlight the versatility of our method, we show how to apply it to explaining predictions and to distinguishing between reliable and non-reliable predictions.

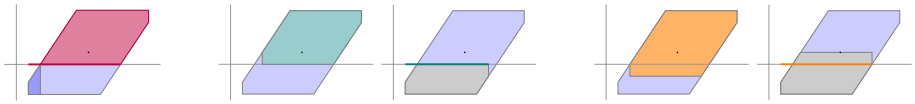
3 Reachable sets of neural networks

The reachable set O w.r.t. an input set I of a neural network f is its output set, i.e. $O = \{f(x) \mid x \in I\}$. Computing the exact reachable set of a neural network is challenging, as proving simple properties of a neural network is already an NP-complete problem (Katz et al. 2017). Under-approximations $\hat{O}_u \subseteq O$ produce a set of points that can definitely be reached with respect to the input, while over-approximations cover all points that might possibly be reached $O \subseteq \hat{O}_o$ (see Fig. 1).

We propose approximating the reachable set by propagating the input set *layer-by-layer* through the neural network. In each layer, the input set is first subjected to the linear transformation defined by weights and biases. This linear transformation is computed *exactly and efficiently* for the zonotope-based set representations we exploit. Then, the ReLU activation function is applied. Since applying ReLU onto a convex set can result in a non-convex set, we approximate convex subsets. Specifically, we propose an analytical solution for the over-approximations and an efficient linear optimization problem formulation for the under-approximations.

Definition of input sets Our approaches operate directly on sets and require an efficient and flexible set representation. For this, we use zonotopes, as they are closed under linear transformation and their G-representation provides a compact representation in high-dimensional spaces. Furthermore, they allow complex and realistic perturbations to be defined that couple input features such as different light conditions on pictures (in short: we go beyond simple and unrealistic norm constraints).

The G-representation of a d-dimensional zonotope $\hat{Z}$ with n generators is defined by a row-vector, the center $\hat{c} \in \mathbb{R}^D$ and a matrix $\hat{G} \in \mathbb{R}^{n \times D}$. The rows of this matrix contain the generators $\hat{g}_i$. The set of all points within $\hat{Z}$ is:



(a) ReLU(Z). **(b)** Under-approximation. **(c)** Over-approximation.

Fig. 2 Application of ReLU to a zonotope (blue) can result in a non-convex set (red). We approximate each subset located in each quadrant separately (here: two quadrants) and subject it to ReLU. The obtained set of sets under-approximates (green sets) or over-approximates (orange sets) ReLU(Z) (Color figure online)

$$\hat{Z} = (\hat{c} \mid \hat{G}) := \left\{ \hat{c} + \sum_{i=1}^{n} \hat{\beta}_i \hat{g}_i \mid \hat{\beta}_i \in [-1, 1] \right\} \subset \mathbb{R}^D. \tag{1}$$

Propagating sets through ReLU networks In this paper we focus on ReLU neural networks, as they are not only widely used but also powerful. A neural network consists of a series of functional transformations, in which each layer l (of n_l neurons) receives the input $x \in \mathbb{R}^{n_{l-1}}$ and produces the output by first subjecting the input to a linear transformation defined by the weights W^l and bias b^l, and then applying ReLU. In the final layer, no activation function is applied, and the output stays in the logit-space. Thus, starting with the *input set* $\hat{Z}_0$ a series of alternating operations is obtained: $\hat{Z}^0 \overset{W^1,b^1}{\rightarrow} Z^1 \overset{\text{ReLU}}{\rightarrow} \hat{Z}^1 \overset{W^2,b^2}{\rightarrow} Z^2 \overset{\text{ReLU}}{\rightarrow} \ldots \overset{W^L,b^L}{\rightarrow} Z^L$, where Z^l denotes the set after the linear transformation, $\hat{Z}^l$ denotes the set after the ReLU, and Z^L is the reachable set (output layer). Since zonotopes are closed under linear transformations, applying weights and bias of layer l to zonotope $\hat{Z}^{l-1} = (\hat{c}^{l-1} \mid \hat{G}^{l-1})$ results in

$$Z^l = \left(c^l \mid G^l\right) = \left(W^l \hat{c}^{l-1} + b^l \mid \hat{G}^{l-1} W^{lT}\right). \tag{2}$$

Obtaining ReLU(Z^l) is challenging, as it may be a non-convex set, as illustrated in Fig. 2a. It is inefficient to further propagate the non-convex set ReLU(Z^l) through the neural network. Therefore, our core idea is to approximate ReLU(Z^l) and use this as the input to the next layer. More precisely, we propose two methods: RsO (reachable set over-approximation) and RsU (reachable set under-approximation). RsO obtains a superset of ReLU(Z^l) in each layer l, while RsU returns a subset. Using RsO within each layer ensures that no points are missed and that the output set captures all reachable points. Equivalently, applying RsU within each layer results in an output set that is a subset of the exact reachable set, i.e. contains the points that will definitely be reached. Pseudocode for RsO, RsU and propagating a zonotope through the neural network is provided in the appendix (see Sect. 6.1).

Approximation of ReLU(Z) In the following, we describe how to approximate ReLU(Z) based on zonotope Z. To unclutter the notiation, we omit layer index l. The ReLU function maps points dimension-wise onto the maximum of themselves and zero. Consideration of dimension d results in three possible cases: Case 1: $\forall p \in Z : p_d < 0$, where the points are mapped to zero, Case 2: $\forall p \in Z : p_d \geq 0$, where the points are mapped onto themselves and Case 3: $\exists p, q \in Z : p_d < 0 \wedge q_d > 0$, where the points are mapped to zero or themselves.

Case 3 causes the non-convexity of ReLU (see Fig. 2a, 2nd dimension). We consider the three cases separately to approximate each maximum convex subset of ReLU(Z) by one zonotope. The three cases are distinguished by computing an index set for each case:

$$\begin{aligned} R_{\mathrm{n}} &= \{d \mid \forall p \in Z : p_d < 0\}, \qquad R_{\mathrm{p}} = \{d \mid \forall p \in Z : p_d \geq 0\}, \\ R &= \{d \mid \exists p, q \in Z : p_d < 0, q_d > 0\} \end{aligned} \tag{3}$$

These index sets can be efficiently obtained through the interval hull of Z (Kühn 1998), where $|.|$ is the element-wise absolute value: $\mathrm{IH}(Z) := \left[c - \delta g, c + \delta g\right]$ where $\delta g = \sum_i |g_i|$. R_{n} contains the dimensions d such that $(c - \delta g)_d \leq 0$, R_{p} contains the dimensions where $(c + \delta g)_d \geq 0$, and R contains the remaining dimensions.

Projection of a zonotope Regarding the dimensions in R_{n}, ReLU maps each point of the zonotope to zero. Thus, we can safely project the whole zonotope $Z = (c \mid G)$ to zero within these dimensions.

Theorem 1 *Let Z be an arbitrary zonotope and $Z' = \mathrm{Proj}_{R_{\mathrm{n}}}(Z)$, then $\mathrm{ReLU}(Z) = \mathrm{ReLU}(Z')$. $\mathrm{Proj}_M(Z) = Z' = \left(c' \mid G'\right)$ is defined by $c'_d = 0$ if $d \in M$, else $c'_d = c_d$ and $g'_{i,d} = 0$ if $d \in M$, else $g'_{i,d} = g_{i,d}$.*

Proof Applying ReLU and the projection operator to Z results in the sets:

$$\begin{aligned} \mathrm{ReLU}(Z) &= \left\{a \mid a_d = \max\{0, b_d\}, b \in Z\right\} \\ \mathrm{Proj}_{R_{\mathrm{n}}}(Z) &= \left\{ q \mid q_d = \begin{cases} 0 \text{ if } d \in R_{\mathrm{n}} \\ p_d \text{ else} \end{cases}, p \in Z \right\} \\ &= \left\{ q \mid q_d = \begin{cases} 0 \text{ if } \forall p \in Z : p_d < 0 \\ p_d \text{ else} \end{cases}, p \in Z \right\} \end{aligned}$$

Applying ReLU on the projection results in $\mathrm{ReLU}(Z)$:

$$\begin{aligned} \mathrm{ReLU}\left(\mathrm{Proj}_{R_{\mathrm{n}}}(Z)\right) &= \left\{a \mid a_d = \max\{0, b_d\}, b \in \mathrm{Proj}_{R_{\mathrm{n}}}(Z)\right\} \\ &= \left\{a \mid a_d = \max\{0, q_d\},\ q_d = \begin{cases} 0 \text{ if } \forall r \in Z : r_d \leq 0 \\ p_d \text{ else} \end{cases}, p \in Z \right\} \\ &= \left\{a \mid a_d = \max\{0, p_d\}, p \in Z\right\} = \mathrm{ReLU}(Z) \end{aligned}$$

□

Projecting Z results in the more compact Z' with no change to the output set. We overload notation, and Z denotes the projected zonotope in the following.

Computation of quadrants that contain a subset S_k of Z Next, we subdivide the projected zonotope Z into subsets located in one quadrant. Quadrants that contain points of Z are determined by an index set R_k, where R_k is an element of the power set $\mathcal{P}(R)$ of R. Each index set R_k corresponds to a set $S_k = \{p \mid p \in Z \wedge p_d \leq 0 \forall d \in R_k \wedge p_d \geq 0 \forall \notin R_k\}$. Clearly, all S_k are convex and disjoint, the union over S_k results in Z. It is important to highlight that we *never materialize the subsets S_k*, as they are unfavorable to compute. Our core idea is to approximate each S_k by zonotope $\hat{Z}_k$. Subsequently, we project $\hat{Z}_k$ in all dimensions of the corresponding R_k (see case 1), resulting in $\mathrm{Proj}_{R_k}(\hat{Z}_k)$. The obtained set of zonotopes is an approximatiuon of $\mathrm{ReLU}(Z)$ and is the input for the next layer. The computation of R, R_k and corresponding subsets S_k is illustrated in the following example.

Example 1 Consider zonotope $Z = (c \mid G)$ (Fig. 2), where $c = (6, 1)$ and generators are $g_1 = (3, 0)$, $g_2 = (2, 3)$ and $g_3 = (0, 0.5)$. The lower bounds are $(1, -2.5)$, the upper bounds are $(11, 4.5)$. As all upper bounds are positive, we do not project any dimension. The index

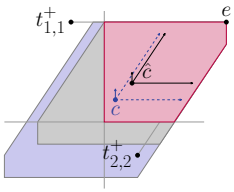


Fig. 3 Overapproximation of S_0 (red) $\subseteq Z$ (blue) by $\hat{Z}$ (gray) (Color figure online)

set considering case 3 is $R = \{2\}$. We need to approximate all subsets S_k corresponding to $R_k \in \mathcal{P}(R)$. The empty set corresponds to the positive quadrant.

We capture each S_k individually to keep the approximations as tight as possible. Theoretically, we could decrease the number of zonotopes by over-approximating several S_k by one zonotope or by not considering small subsets S_k in the case of under-approximation. We discuss such an extension that restricts the maximum number of zonotopes at the end of this section. This extension enables a balance between tightness of approximations and run-time, which is useful for larger neural networks. The approximation of S_k can be either an over-approximation (RsO) or an under-approximation (RsU).

Over-approximation of ReLU(Z) Given $S_k \subseteq Z$, we aim to over-approximate S_k by $\hat{Z}_k = (\hat{c} \mid \hat{G})$ (to unclutter the notation, we omit the index k w.r.t. center and generators). Our core idea is that if $\hat{Z}_k$ is a tight over-approximation of S_k, the shape of $\hat{Z}_k$ should resemble the shape of Z (see Fig. 2c).

As the shapes of two zonotopes are similar if their generators point in similar directions, we derive $\hat{Z}_k$ from the generators of Z. More precisely, the generators of $\hat{Z}_k$ are obtained by scaling each generator g_j of Z with a factor $\alpha_j \in [0, 1]$ and computing the shift of the center such that $S_k \subseteq \hat{Z}_k \subseteq Z$. Clearly, $\alpha_j = 1$ fulfills this property, but results in $\hat{g}_j = g$ and a loose over-approximation. Thus, we aim to minimize over α_j. Each scaling factor α_j for generator g_j is computed analytically (see Fig. 3) by first computing an extreme point e of the zonotope. We start in e and test if the generator g_j allows a point $t_{j,d}$ to be reached outside the quadrant under consideration. If this is the case, g_j can be scaled down and $\hat{Z}$ still over-approximates S_k. We compute the extreme points and scaling factors for each dimension d, resulting in $\alpha_{j,d}$.

Regarding dimension d, g_j can be scaled by the factor $\alpha_{j,d}$. If we scale g_j by a larger factor, we leave the quadrant corresponding to S_k with respect to dimension d. A larger scaling factor is not necessary in order to over-approximate S_k. Thus, we minimize over $\alpha_{j,d}$ to obtain the smallest α_j and the tightest over-approximation. Formally, $\hat{g}_j$ and $\hat{c}$ of the over-approximating zonotope $\hat{Z}_k$ are:

$$\hat{g}_j = \alpha_j g_j, \quad \hat{c} = c + \sum_j s_j(1-\alpha_j) o_j g_j \quad \text{with the following definitions:}$$

$$\alpha_j = \min_d \alpha_{j,d}, \quad d^* = \arg\min_d \alpha_{j,d}, \quad o_j = \frac{g_{j,d^*}}{\left|g_{j,d^*}\right|}, \quad s_j = 1 \text{ if } d^* \notin R_k, -1 \text{ else}$$

$$\forall d \notin R_k : t^+_{j,d} = c_d - 2\left|g_{j,d}\right| + \sum_i \left|g_{i,d}\right|, \alpha_{j,d} = 1 - \frac{|t^+_{j,d}|}{2|g_{j,d}|} \text{ if } t^+_{j,d} < 0, \ 1 \text{ else} \tag{4}$$

$$\forall d \in R_k : t^-_{j,d} = c_d + 2\left|g_{j,d}\right| - \sum_i \left|g_{i,d}\right|, \alpha_{j,d} = 1 - \frac{|t^-_{j,d}|}{2|g_{j,d}|} \text{ if } t^-_{j,d} > 0, \ 1 \text{ else}$$

Although the generators of Z are scaled down, the obtained zonotope $\hat{Z}$ is an over-approximation of S_k for the respective quadrant (which we never computed explicitly). This is shown in Theorem 2 by using Lemma 3 and 4.

Theorem 2 *Let* $Z = (c \mid G)$, $S_k = \{p \mid p \in Z \wedge p_d \geq 0\ \forall d \notin R_k \wedge p_d \leq 0\ \forall d \in R_k\}$ *and* $\hat{Z}_k = (\hat{c} \mid \hat{G})$ *with the center and generators as defined in Equation 4. Then* $S_k \subseteq \hat{Z}_k$.

Proof Let $p \in S_k$. Since $p \in Z$ it exists $\beta_j \in [-1, 1]$ such that:

$$\begin{aligned} p =& c + \sum_j \beta_j g_j = \hat{c} - \sum_j s_j(1-\alpha_j)o_j g_j + \sum_j \beta'_j o_j g_j \\ =& \hat{c} + \sum_j (\beta'_j - s_j(1-\alpha_j))o_j g_j = \hat{c} + \sum_j \frac{\beta'_j - s_j(1-\alpha_j)}{\alpha_j} o_j \hat{g}_j \end{aligned} \tag{5}$$

where we use that $o_j = \frac{g_{j,d^*}}{|g_{j,d^*}|} \in \{-1, 1\}$, $\beta'_j = o_j\beta_j$, $c = \hat{c} - \sum_j s_j(1-\alpha_j)o_j g_j$ and $g_j = \frac{1}{\alpha_i}\hat{g}_j$. If we can show that $\frac{\beta'_j - s_j(1-\alpha_j)}{\alpha_j} o_j \in [-1, 1]$ then $p \in Z_k$. To this end, we distinguish how α_j is obtained: If α_j is computed based on $d^* \notin R_k$ then $s_j = 1$ and it holds that $2(1-\alpha_j) - 1 \leq \beta'_j$ (see Lemma 3). If α_j is computed based on $d^* \in R_k$ then $s_j = -1$ and $\beta'_j \leq 1 - 2(1-\alpha_j)$ (see Lemma 4). With these constraints and $\beta'_j \in [-1, 1]$ (from the definition of zonotopes) we obtain $\frac{\beta'_j - s_j(1-\alpha_j)}{\alpha_j} \in [-1, 1]$. Using $o_j \in \{-1, 1\}$, we define $\hat{\beta}_j = \frac{\beta'_j - s_j(1-\alpha_j)}{\alpha_j} o_j$ and obtain $p = \hat{c} + \sum_j \hat{\beta}_j \hat{g}_j \in \hat{Z}_k \Rightarrow \forall p \in S_k : p \in \hat{Z}_k$ and $S_k \subseteq \hat{Z}_k$. □

Lemma 3 *Consider zonotope* $Z = (c \mid G)$. *Let* $S_k = \{p \mid p \in Z \wedge p_d \geq 0\ \forall d \notin R_k \wedge p_d \leq 0\ \forall d \in R_k\}$ *and let* $\hat{Z}_k$ *be a zonotope with the center and generators defined in Equation 4. Consider the definitions used in Theorem 2. Then* $2(1-\alpha_j) - 1 \leq \beta'_j$ *if* α_j *corresponds to a* $d \notin R_k$.

Proof We use that for a point $p \in Z : p_d \geq 0$ in case $d \notin R_k$ and $t^+_{j,d} < 0$.

$$0 \leq p_d \tag{6}$$

$$\Leftrightarrow t^+_{j,d} + 2\frac{1}{2}\frac{|t^+_{j,d}|}{|g_{j,d}|}|g_{j,d}| \leq c_d + \sum_i \beta_i g_{i,d} \tag{7}$$

$$\Leftrightarrow t^+_{j,d} + 2(1-\alpha_j)|g_{j,d}| \leq c_d + \sum_i \beta_i g_{i,d} \tag{8}$$

$$\Leftrightarrow c_d + \sum_i |g_{i,d}| - 2\left|g_{j,d}\right| + 2(1-\alpha_j)|g_{j,d}| \leq c_d + \sum_i \beta_i g_{i,d} \tag{9}$$

$$\Leftrightarrow \sum_{i,i\neq j} o_{i,d} g_{i,d} - |g^d_j| + 2(1-\alpha_j)|g_{j,d}| \leq \sum_{i,i\neq j} \beta_i g_{i,d} + \beta_j g_{j,d} \tag{10}$$

$$\Leftrightarrow \sum_{i,i\neq j}(o_i^d - \beta_i)g_{i,d} + (2(1-\alpha_j)-1)|g_{j,d}| \leq \beta'_j|g_{j,d}| \tag{11}$$

$$\Leftrightarrow \underbrace{\frac{1}{|g_{j,d}|}\sum_{i,i\neq j}(o_{i,d}-\beta_i)g_{i,d}}_{\geq 0} + (2(1-\alpha_j)-1) \leq \beta'_j \tag{12}$$

$$\Rightarrow 2(1-\alpha_j)-1 \leq \beta'_j \tag{13}$$

We use that $p_d = c_d + \sum_i \beta_i g_{i,d}$ (7), from the definitions of $t^+_{j,d}$ $1-\alpha_{j,d} = \frac{|t^+_{j,d}|}{2|g_{j,d}|}$ if $t^+_{j,d} < 0$ and 0 else (8), the definition of $t^+_{j,d}$ (9), $o_{j,d}g_{j,d} = \left|g_{j,d}\right|$ (10) and $\beta_j g_{j,d} = \beta'_j|g_{j,d}|$ (11). Inequality $\frac{1}{|g_{j,d}|}\sum_{i,i\neq j}(o_{i,d}-\beta_i)g_{i,d} \geq 0$ (12) holds because $o_{i,d}g_{i,d} = |g_{i,d}|$ and $\beta_i g_{i,d} = \pm\beta_i|g_{i,d}| \leq |g_{i,d}|$ because $\beta_i \in [-1,1] \Rightarrow$ $(o_{i,d}-\beta_i)g_{i,d} \geq 0$ and thus, $\sum_{i,i\neq j}(o_{i,d}-\beta_i)g_{i,d} \geq 0$. □

Lemma 4 *Consider zonotope* $Z = (c \mid G)$. *Let* $S_k = \{p \mid p \in S_k = \{p \mid p \in Z \wedge p_d \geq 0\ \forall d \notin R_k \wedge p_d \leq 0 \forall d \in R_k\}$ *and let* $\hat{Z}_k$ *be a zonotope with the center and generators defined in Equation 4. Consider the definitions used in Theorem 2. Then* $\beta'_j \leq 1-2(1-\alpha_j)$ *if* α_j *corresponds to a* $d \in R_k$.

The proof of Lemma 4 is similar to the one of Lemma 3 and not shown in detail. The differences to the previous proof are that for a point $p \in Z : p_d \leq 0$ in case $d \in R_k$ and $t^-_{j,d} > 0$, we start with $0 \geq p_d$, we use the $t^-_{j,d}$ instead of $t^+_{j,d}$ and signs of the terms are different.

The subset S_k is located in one quadrant and the corresponding R_k contains dimensions that are mapped to zero by ReLU (case 1 on S_k). Thus, we project the over-approximation $\hat{Z}_k$ in dimensions $d \in R_k$ as described above. This projection is exact: $\text{ReLU}(S_k) = \text{Proj}_{R_k}(S_k) \subseteq \text{Proj}_{R_k}(\hat{Z}_k)$.

Under-approximation of ReLU(Z) Finding a tight under-approximation of S_k turns out to be more challenging. We propose to tackle this by solving a constrained optimization problem, in which we aim to find a zonotope $\hat{Z}_k$ of maximum volume subject to the constraint $\hat{Z}_k \subseteq S_k$:

$$\hat{Z}_k = \arg\max_{\hat{Z}} V(\hat{Z}) \ \text{ subject to } \ \hat{Z} \subseteq S_k \tag{14}$$

How can we instantiate Equation 14 to under-approximate S_k tightly and keep computations efficient? We derive an efficient linear program by considering the same search domain as before. More precisely, we constrain the search space to zonotopes that are derived from the original zonotope Z, by scaling its generators g_i by factors $\alpha_i \in [0,1]$, i.e. $\hat{g}_i = \alpha_i g_i$. As motivated before, this assumption is reasonable, since $\hat{Z}_k$ and Z have similar shapes.

Importantly, to ensure that we under-approximate a part of Z located in one quadrant, we add a constraint that forces the lower bound of the interval hull of $\hat{Z}$ to be non-negative if $d \notin R_k$: $\hat{c}_d - \sum_i \left|\hat{g}_{i,d}\right| \geq 0 \ \ \forall d \notin R_k$ and one that forces the upper bound of the interval hull of $\hat{Z}$ to be negative if $d \in R_k$: $\hat{c}_d + \sum_i \left|\hat{g}_{i,d}\right| \leq 0 \ \ \forall d \in R_k$. Since the volume of the zonotope grows with α_i, we instantiate the objective function by $\sum_i \alpha_i$. Combining all of these considerations results in the following linear optimization problem:

$$\alpha^*, \delta^* = \arg\max_{\alpha,\delta} \sum_i \alpha_i \text{ subject to } \hat{g}_i = \alpha_i g_i, \quad \alpha_i \in [0,1],$$
$$\hat{c} = c + \sum_i \delta_i g_i, \quad |\delta_i| \leq 1 - \alpha_i$$
$$\hat{c}_d - \sum_i |\hat{g}_{i,d}| \geq 0 \; \forall d \notin R_k, \quad \hat{c}_d + \sum_i |\hat{g}_{i,d}| \leq 0 \; \forall d \in R_k$$

Theorem 3 *Let $\hat{Z}_k$ be computed from zonotope Z based on α^*, δ^*, then $\hat{Z}_k \subseteq S_k$.*

Proof Let $\gamma_i = \frac{\delta_i}{1-\alpha_i}$. Since $|\delta_i| \leq 1 - \alpha_i$ it holds that $\gamma_i \in [-1, 1]$. Since $p \in \hat{Z}_k$ $\hat{\beta}_i \in [-1, 1]$ exists:

$$p = \hat{c} + \sum_i \hat{\beta}_i \hat{g}_i = c + \sum_i \delta_i g_i + \sum_i \hat{\beta}_i \alpha_i g_i$$
$$= c + \sum_i \gamma_i (1 - \alpha_i) g_i + \sum_i \hat{\beta}_i \alpha_i g_i = c + \sum_i \big((1 - \alpha_i)\gamma_i + \alpha_i \hat{\beta}_i\big) g_i$$

To prove that $p \in Z$ we need to show that $(1 - \alpha_i)\gamma_i + \alpha_i \hat{\beta}_i \in [-1, 1]$. Considering $\gamma_i \in [-1, 1]$, $\alpha_i \in [0, 1]$ and $\hat{\beta}_i \in [-1, 1]$ we obtain:

$$\forall i : (1 - \alpha_i)\gamma_i + \alpha_i \hat{\beta}_i \geq -(1 - \alpha_i) - \alpha_i = -1$$
$$\forall i : (1 - \alpha_i)\gamma_i + \alpha_i \hat{\beta}_i \leq (1 - \alpha_i) + \alpha_i = 1$$

Thus, we define $\beta_i = (1 - \alpha_i)\gamma_i + \alpha_i \hat{\beta}_i$ and obtain $p = c + \sum_i \beta_i g_i$ and thus, $\hat{Z}_k \subseteq Z$. The constraints $\hat{c}_d - \sum_i |\hat{g}_{i,d}| \geq 0 \forall d \notin R_k$ and $\hat{c}_d + \sum_i |\hat{g}_{i,d}| \leq 0 \forall d \in R_k$ ensure that $\hat{Z}$ is located in the desired quadrant:

$$p_d = \hat{c}_d + \sum_i \hat{\beta}_i \hat{g_{i,d}} \geq \hat{c}_d - \sum_i |\hat{g}_{i,d}| \geq 0 \text{ if } d \notin R_k$$
$$p_d = \hat{c}_d + \sum_i \hat{\beta}_i \hat{g_{i,d}} \leq \hat{c}_d + \sum_i |\hat{g}_{i,d}| \leq 0 \text{ if } d \in R_k$$

Thus, $\hat{Z}_k \subseteq S_k$. □

If the quadrant under consideration is empty (which can happen for many quadrants) the optimization problem is not solvable and we can safely ignore this quadrant. Since all points in $\hat{Z}_k$ are negative w.r.t. the dimensions $d \in R_k$ (case 1), we compute $\text{Proj}_{R_k}(\hat{Z}_k)$ and obtain an under-approximation of $\text{ReLU}(S_k)$. See Fig. 2b for illustration.

Balancing approximation tightness and run-time For large input sets, the number of convex subsets that define the reachable set of a neural network scales exponentially with the number of neurons. Let us consider zonotope $Z = (c \mid G)$, $c \in \mathbb{R}^D$, $G \in \mathbb{R}^{n \times D}$. In the worst case, Z consists of points that are spread over 2^D quadrants. RsO and RsU approximate each subset S_k located in one quadrant by a separate zonotope Z_k.

To balance approximation tightness and run-time, we extend RsO and RsU, such that the number of zonotopes can be restricted by the user. The overall number of zonotopes (w.r.t. the whole neural network) is restricted by B and the amplification is restricted by A. The amplification is the maximum number of zonotopes Z_k used to approximate $\text{ReLU}(Z)$ w.r.t. one layer and one input zonotope Z. It is defined by the number of quadrants q that

contain points of Z and can be computed as follows. First, we compute the interval hull of Z. With respect to dimension d, all points within Z are in the interval $\left[l^d_{\text{low}}, l^d_{\text{upp}}\right] = \left[c_d - \delta g_d, c_d + \delta g_d\right]$, where $\delta g = \sum_i |g_i|$. Second, we count the number R_n of dimensions d where $l^d_{\text{low}} < 0$ and $l^d_{\text{upp}} > 0$. The number of quadrants is $q = 2^{R_n}$.

Over-approximation: If $q > A$, we compute the interval hull of Z and restrict the intervals to their positive portion. Thus, we over-approximate ReLU(Z) by one zonotope instead of q zonotopes. If the overall number of zonotopes is larger than B, we estimate the size of each zonotope $Z = (c \mid G)$ by $\text{size}(Z) = \sum_d \log(\delta g)_d$, where $\delta g = \sum_i |g_i|$. The largest $B - 1$ zonotopes are kept while the smaller ones are merged (i.e. we compute an over-approximation of their union by minimizing/maximizing over the lower/upper limits of their interval hulls). The resulting interval is transformed into the G-representation of a zonotope: $l_{\text{low}} = \min_{Z_k}(c_k - (\delta g)_k)$, $l_{\text{upp}} = \max_{Z_k}(c_k + (\delta g)_k)$, $Z_{\text{uni}} = (l_{\text{low}} + g_{\text{ext}} \mid \text{diag}(g_{\text{ext}}))$, $g_{\text{ext}} = 0.5(l_{\text{upp}} - l_{\text{low}})$ where $\text{diag}(g_{\text{ext}})$ is a diagonal matrix.

Under-approximation: In this case, we simply drop the smallest zonotopes if $q > A$ or the overall number of zonotopes is larger than B.

4 Applications and experiments

We highlight the versatility of our RsO and RsU approach by describing several applications in classification and regression tasks. More specifically, we discuss (non-)robustness verification, robust training, quantification of feature importance and the distinction between reliable and non-reliable predictions. Furthermore, we analyze reachable sets of an autoencoder. Our input zonotopes capture three different shapes: cube (equivalent to L_∞ -norm), box (with a different perturbation on each feature) and free (with coupling of features). We train feed-forward ReLU networks on standard data sets.

Experimental setup Our approaches are implemented in Python/Pytorch. We train feed-forward ReLU networks using stochastic gradient descent with cross-entropy loss (classifiers), Huber loss (regression models), mean-square-error loss (autoencoder models) or robust loss functions (see following sections) and early stopping. Experiments are carried out on the following popular data sets and neural network architectures (accuracy denotes worst accuracy obtained for this data set by one of the specified neural network architectures): **Classifiers**: *Iris* (Fisher 1936; Dua and Graff 2017): 3 classes, 4 features, $1-5$ hidden layers of 4 neurons each. *Wine* (Forina et al. 1990; Dua and Graff 2017): 3 classes (cultivars), 13 features, $1-5$ hidden layers of 6 neurons each. *Tissue* (Jossinet 1996; Dua and Graff 2017): breast tissue probes, 6 classes, 9 features, $1-3$ hidden layers of 8 neurons each. Breast *cancer* Wisconsin (diagnostic) (Street et al. 1999; Dua and Graff 2017): 2 classes, 30 features, $1-2$ hidden layers of 10 neurons each. *MNIST* (LeCun et al. 2010): 28×28 gray-scale images, 10 classes, 1 hidden layer of 15 neurons. *Fashion-MNIST* (Xiao et al. 2017): 28×28 gray-scale images, 10 classes, 1 hidden layer of 15 neurons or 5 hidden layers of 30 neurons. **Regression Models**: *Abalone* (Nash et al. 1994): 8 features, 1 output, $1-3$ hidden layers of 6 neurons each. *Housing* (Harrison and Rubinfeld 1978): 13 features, 1 output, 1 hidden layer of 13 neurons. *Airfoil* (Brooks et al. 1989): 5 features, 1 output, $1-4$ hidden layers of 5 neurons each. **Autoencoder**: *MNIST* (LeCun et al. 2010): 28×28 gray scale images, 28×28 output, 3 hidden layers of $30 \times 60 \times 30$ neurons. *Fashion-MNIST* (Xiao et al. 2017): 28×28 gray scale images, 28×28 output, 3 hidden layers of $30 \times 60 \times 30$ neurons.

For classification, all data sets are balanced by sub-sampling training- and test-sets such that evaluation experiments are done on the same amount of points for each class. The input size of MNIST and Fashion-MNIST is reduced from 28×28 to 30 by using principle component analysis (PCA). In the evaluation experiments, we use 30 input points of the iris data set, wine data set and tissue data set, 86 points of the cancer data set, 200 point of the MNIST data set and 100 points of the Fashion-MNIST data set, which are not part of the training set.

Experiments are conducted in Python (version 3.6) on a machine with 10 Intel Xeon CPU cores with 2.2 GHz, 4 GEFORCE GTX 1080 Ti and 256 GB of RAM running Ubuntu (version 16.04.6).

Definition of input sets Using zonotopes as input sets has the advantage that we are able to verify different kinds of perturbations. Here, the input set $\hat{Z} = (\hat{c} \mid \hat{G})$ is defined by using an input data point x as center $\hat{c}$ and the following perturbations specified by the generator matrix $\hat{G}$. *Cube*: $\hat{Z}_{\text{cube}}$ is a hyper-cube whose shape is equivalent to the unit ball of the L_∞ -norm. As the allowed perturbation on each input feature is the same, the generator matrix is εI_d for different ε. *Box* $\hat{Z}_{\text{box}}$ is a so called axis-aligned parallelotope (n-dimensional box). This shape allows different disturbances on each input feature, but it does not couple features. For this, we first compute a zonotope by using the eigen-vectors that correspond to the d largest eigenvalues of the data set as generators. Z_{box} is obtained by computing the interval hull of this zonotope and scaling its volume such that it is equivalent to the volume of $\hat{Z}_{\text{cube}}$ for a given ε. *Free* $\hat{Z}_{\text{free}}$ is an arbitrary zonotope that enables disturbances to be coupled between input features which cannot be captured by norms or intervals. This input zonotope is obtained by increasing/decreasing all feature values simultaneously by at most ε and additionally allowing a small, fixed perturbation $\delta \ll \varepsilon$ on each feature. If the input is an image, this perturbation would brighten/darken all pixel values simultaneously: $\hat{G} = [\delta I_d, \varepsilon \mathbf{1}]$.

For feature rankings, the following setting is used: to quantify the influence of feature f_1 on the prediction of x, we define a box-shaped input set $\hat{Z}_{f_1} = (x|G)$ around x that allows a perturbation δ on f_1 and a minimal perturbation ε (here: $\varepsilon = 0.01$) on all other features. More formally, we use a diagonal input matrix G, where $G_{1,1} = \delta$ and $G_{i,i} = \varepsilon \; \forall i \neq 1$.

Classification: (Non-)robustness verification First, we evaluate the potential of reachable sets by using them for robustness/non-robustness verification, i.e. for studying how predictions of a classifier change when perturbing input instances. More precisely, we aim to analyze if predictions based on an input set map to the same class or if they vary. Formally, the set of predictions (classes) is $P = \{\arg\max_c f(x)_c | x \in I\}$, given input set I.

For verification, we compute a robustness score against each class. Let a be the predicted class and $b \neq a$ the class against which we quantify robustness.[1] The least robust point p within the reachable set (output/logit space) is the one where its coordinate p_b is close to or larger than p_a. Based on these considerations, we define the robustness score against class b of reachable set R_S:

$$\mathrm{s}_b = \min_{p \in R_S} \left(p_a - p_b\right) = \min_{Z=(c|G) \in R_S} \left(c_a - c_b - \sum_i \left| g_i^a - g_i^b \right| \right) \tag{15}$$

[1] Please note that reachable sets capture all classes jointly. More precisely, the method does not require any label/class information at all. Thus, it is directly applicable to other tasks such as regression.

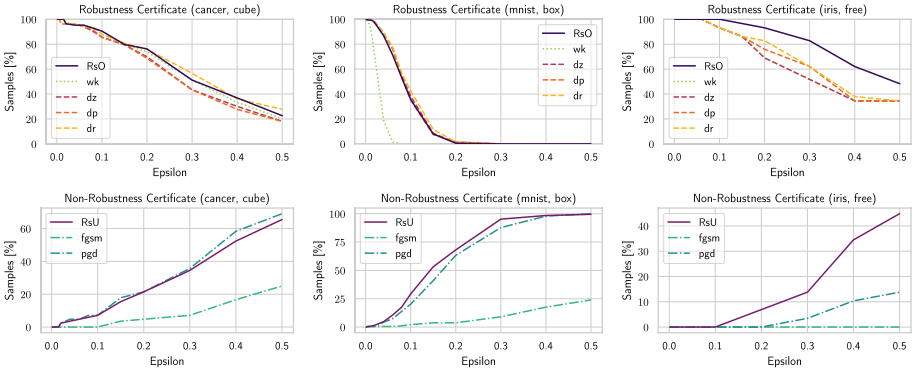


Fig. 4 Performance evaluation of RsO and RsU in (non-)robustness verification on the cancer data-set (left, 2 hidden layers, acc. 97%), MNIST (middle, 1 hidden layer, acc. 94%) and iris (right, 5 hidden layers, acc. 97%)

where $Z \in R_S$ denotes the computed zonotopes, and we use that $p_a = c_a + \sum_i \beta_i g_i^a$, $p_b = c_b + \sum_i \beta_i g_i^b$ and $\sum_i \beta_i (g_i^a - g_i^b)$ is minimal if $\beta_i \in \{-1, 1\}$ depending on the sign of $g_i^a - g_i^b$.

Robustness certificates are obtained by computing the scores against all classes $b \neq a$ on the *over*-approximated reachable set R_{SO}. If *all* scores are *positive*, the robustness certificate holds, and all points from the input set are classified as class a. Non-robustness certificates are obtained by checking if there is a class b, such that s_b on the *under*-approximated reachable set R_{SU} is *negative*. If this is the case, at least one point from the input set is categorized as class b.

There are three benefits to these scores. First, computing scores is efficient (see Equation 15). What is more, the scores are fully differentiable w.r.t. the network parameters, enabling immediate robust training (see later experiment). Second, the scores are applicable to *class-specific verification* (i.e. robust against class b_1, non-robust against b_2). And thirdly, the scores allow relative quantification of (non-)robustness. A reachable set with a high score is more robust than one with a low score.

We compare the performance of RsO on robustness verification using the state-of-the-art methods, Wong and Kolter (2018), Singh et al. (2018), Singh et al. (2019b), Singh et al. (2019c) and es (exact approach) (Xiang et al. 2017), which computes the exact reachable set (implementation (Liu et al. 2019)). RsU is compared with the success rate of FGSM attacks (Goodfellow et al. 2015; Szegedy et al. 2014) and PGD attacks (Madry et al. 2018). To handle the box setting, FGSM attacks are scaled, such that the perturbed input is contained within the input zonotope. The PGD attack is projected onto the input zonotope in each step, i.e. extended to handle arbitrary input zonotopes. Figures 4 and 5 illustrate (non-)robustness verification on the cancer data set, MNIST, iris data set and FashionMNIST for cube-, box- and free-shaped input zonotopes.For robustness verification, we measure the number of samples for which the scores against all non-target classes are positive. For non-robustness verification, we count the number of samples in which a negative score exists against a class. In the cube and box settings, RsO perform similar way to dr, dz and dp, while RsU is similar (cube) or slightly better (box) than PGD attacks. Based on arbitrary input zonotopes (free setting), RsO and RsU outperform both state-of-the-art robustness verification approaches and PGD attacks.

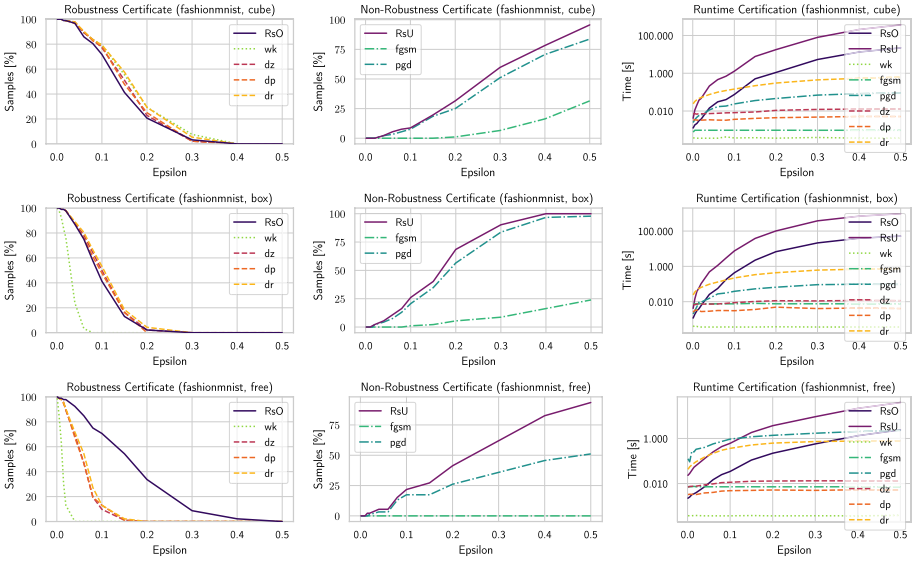


Fig. 5 Performance evaluation and run-time of RsO and RsU in (non-)robustness verification on Fashion-MNIST (acc. 92%) using cube-shaped (top), box-shaped (middle) and freely-shaped input sets (bottom)

Table 1 Comparison of RsO and RsU with the exact reachable set computation (es) (Xiang et al. 2017; Liu et al. 2019) on 29 correctly classified samples of the iris data set (neural network with 1 hidden layer of 4 neurons, acc. 97%, cube setting)

ϵ	RsO		RsU		es		
	No. rob.	Time [ms]	No. non-r.	Time [ms]	No. rob.	No. non-r.	Time [ms]
0.001	29	0.47	0	0.46	28	0	14.68
0.005	29	0.46	0	0.47	28	0	14.56
0.01	29	0.47	0	0.46	28	0	14.61
0.02	29	0.58	0	2.40	–	–	> 3d

The run-time of RsO and RsU increases with the number of input features, the number of neurons in the neural network and the perturbation ϵ. The dependency on ϵ is due to the fact that huge sets usually decompose into more convex subsets than smaller sets when they are subject to ReLU, and so, run-time increases with the size of the input set. Note that we compute the full reachable set of the neural network, which provides much more information than a binary (non-)robustness-certificate. The other techniques, dz, dp, dr are designed for robustness verification/attacks and do not return any further information. A run-time comparison is thus biased. Still, for smaller ϵ and also for the free-shaped input, the absolute run-time of our methods is competitive.

Since es (Xiang et al. 2017; Liu et al. 2019), which computes the exact reachable set, requires too much time even with the smallest neural network architecture, it was not possible to conduct a meaningful comparison. The exact approach es only ran on the smallest neural network (iris data set, neural network with 1 hidden layer of 4 neurons) for the

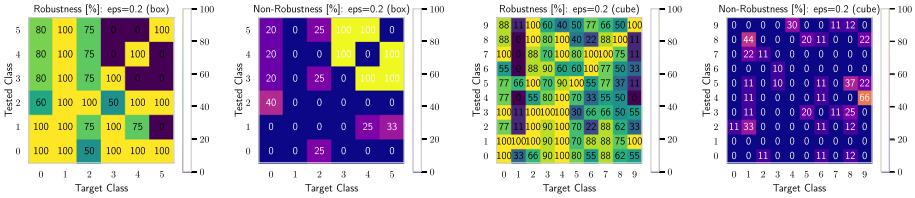


Fig. 6 Class specific verification on the breasttissue data-set (3 hidden layer, acc. 97%, box setting, left) and Fashion-MNIST (1 hidden layer, acc. 92%, cube setting, right)

Table 2 Distinguishing between reliable and non-reliable predictions: comparison of RsO and softmax scores (fashionmnist, classification acc. 96 %, $\epsilon = 0.005$)

	TPR [%]	TNR [%]	reliablity acc[%]
RsO	90.5	75.0	89.5
softmax scores	92.0	71.4	90.7

smallest perturbations $\epsilon \in \{0.001, 0.005, 0.01\}$ (see Table 1)[2]. Note that the exact approach es certifies 28 of the 29 samples as robust and 0 as non-robust and rejects one sample for which it was not able to solve an underlying optimization problem. When performing the exact method es on a cube-shape input with perturbation $\epsilon = 0.02$, it did not finish even after more than three days. This might be explained by the fact that es uses half-spaces to describe the reachable set. Applying ReLU on sets described by half-spaces requires exponential time, and thus, es is not feasible even for small neural networks. Consequently, the reachable set needs to be over-/under-approximated as in our approach.

Classification: class-specific verification Robustness scores allow class-specific (non-) robustness verification in cases where distinguishing between classes is not equally important, e.g. in the tissue data set. The authors of the data set are of the opinion that distinguishing between the class 3, 4 and 5 (fibro-adenoma, mastopathy and glandular) is of minor importance, while it is crucial to distinguish these classes from class 1 (carcinoma). This is illustrated in Fig. 6, left part, where classes 3, 4 and 5 are not robust against each other, while class 1 is robust against all other classes (plot: percentage of instances which are evaluated as (non-)robust; x-axis: ground truth class, y-axis: class we test against). Thus, class-specific analysis allows classifiers to be evaluated more specifically and focus on crucial robustness properties.

Furthermore, it allows us to draw conclusions about the concepts a neural network has learned (see Fig. 6, right part, Fashion-MNIST with classes: 0 top, 1 trousers, 2 pullover, 3 dress, 4 coat, 5 sandal, 6 shirt 7 sneaker, 8 bag, 9 boot). It is striking that class 2 pullover is less robust against classes of items of a similar shape (0 top, 4 coat) but robust against classes of items of different shapes (1 trousers, 3 dress, 5 sandal, 8 bag, 9 boot).

[2] Note that we used a version of Liu et al. (2019) in which a previously existing bug in an underlying library has been fixed. This fix is crucial for correctness, but results in longer run-times than originally reported in (Liu et al. 2019; Xiang et al. 2017).

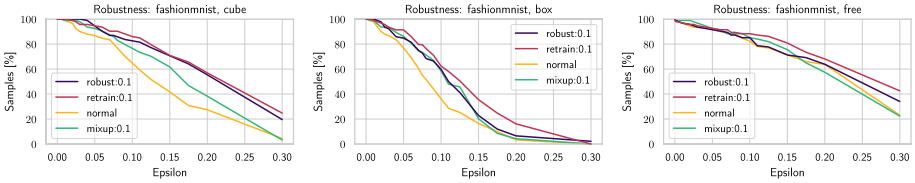


Fig. 7 Evaluation of gg against box-shaped perturbations with $\epsilon = 0.1$ on FashionMNIST (Acc.: normal 91%, mixup 94%, retrain 93%, robust 92%)

This indicates that the neural network has extracted the shape and learned its importance for a classification decision.

Classification: reliability of predictions Distinguishing between reliable (label 0) and non-reliable predictions (label 1) can be seen as a binary classification problem. Although a wrong prediction (w.r.t. ground truth) can theoretically have a high robustness score, we observe that the robustness scores corresponding to wrongly predicted inputs are mostly negative or close to zero. Thus, we consider a prediction as reliable if the corresponding robustness scores (w.r.t. the predicted class) is larger than a positive threshold θ. This threshold θ is chosen such that it maximizing the number of correctly identified reliable/ non-reliable samples on the validation set. Table 2 compares the performance of RsO with our proposed baseline approach that uses softmax scores to distinguish between reliable and non-reliable predictions.

Our comparison shows that, while softmax scores result in a slightly higher true-positive-rate and overall accuracy, RsO provides a significantly higher true-negative-rate. Thus, RsO identifies more non-reliable predictions than softmax scores. Furthermore, RsO provides a robustness certificate as well as an indicator for reliability.

Classification: robust training The robustness scores as defined in Equation 15 are directly used in robust training by incorporating them into the loss function, e.g. as follows:

$$L_{\text{rob}} = L_{\text{pred}} + \mathbb{I}[\text{pred=target}] \cdot \max_b \text{ReLU}\left(-s_b\right) \tag{16}$$

where L_{pred} is the cross-entropy loss and $\mathbb{I}[\text{pred=target}] = 1$ for correctly classified inputs, otherwise 0. Note that the loss is fully differentiable w.r.t. the neural network weights (i.e. we can backpropagate through the zonotope construction) which makes it possible to train a model with enhanced robustness against any perturbation that can be described by any (input) zonotope. Figure 7 compares robustness of models obtained by robust training (L_{rob}), retraining (warm-start with a normally trained model, further training with L_{rob}), normal training, and mixup (a robust training technique based on a convex combination of samples, see Zhang et al. 2018).

Robust training, retraining and mixup enhance the robustness of the neural network on cube-, box- and free-shaped perturbations as well as the accuracy of the neural network. While the performance of mixup and robust training are comparable on box- and free-shaped perturbations, retraining outperforms mixup on all three perturbation shapes.

Regression: (Non-)robustness analysis and robust training Obtaining robust neural networks is desirable in any task but has mainly been studied for the purpose of classification. Classifiers are robust if an input x and all points in its neighborhood are assigned to the same label. In regression tasks, there is no equivalent robustness definition, because outputs are continuous and not categorical. However, intuitively, regression models are robust

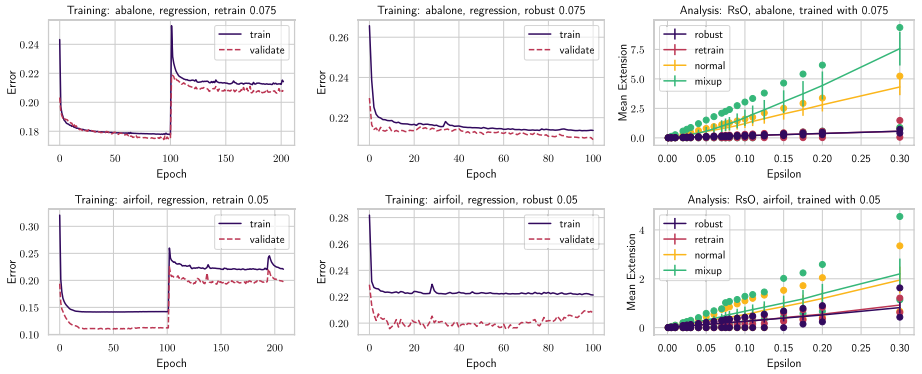


Fig. 8 Robust training and robustness analysis of regression models. The smaller the mean extension the more robust is the model. (Error on the test set—abalone data: normal 0.20, retrain 0.24 (start at epoch 100), robust 0.24, mixup 0.20, airfoil data: normal 0.12, retrain 0.18, robust 0.18, mixup 0.11)

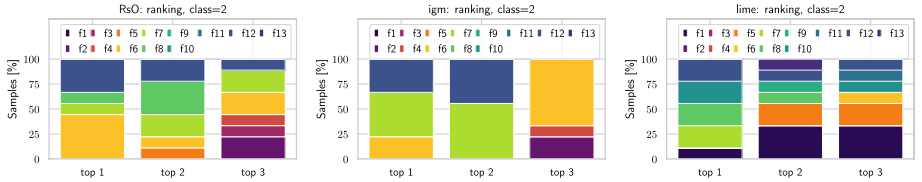


Fig. 9 Ranking (top 3 features) for samples (y-axis) of the wine data set (class 2, 13 features, 1 hidden layer, neural network accuracy 93.3 %) computed by RsO (left), igm (middle) and lime (right)

if close inputs result in close outputs. Assume that inputs and outputs are standardized before training, such that all features are on an equal scale. The extension l_a of output feature a within the reachable set R_S quantifies robustness: the smaller l_a is, the more robust is the model. The extension is defined by the two most distant points u and v within R_S w.r.t. dimension a: $l_a = \left|\max_{u \in R_S} u_a - \min_{v \in R_S} v_a\right|$. For input features, the extension l_{in} is equivalently defined on the input set. In the cube setting, l_{in} is the same for all input features.

If we have $l_a \leq l_{\text{in}}$ for all output features a, the regression model maps close inputs to close outputs and we consider it as robust. We use this robustness definition to define a robust training function based on feature extension and a standard loss function L_{val} (e.g. Huber loss):

$$L_{\text{rob}} = L_{\text{val}} + \text{ReLU}\left(\max_a l_a - l_{\text{in}}\right) \tag{17}$$

If l_a is larger than l_{in} the second term of L_{rob} is positive, otherwise it is zero. We compare four different training modes: normal (training with Huber loss), retrain (warm-start with a normally trained model, and further training with L_{rob}), robust (training with L_{rob}), and mixup (a training technique that convexly combines inputs, see Zhang et al. 2018). Figure 8 illustrates the training and robustness analysis, based on the abalone data set (2 hidden layers, first row) and the airfoil dataset (1 hidden layer, second row).

While mixup seems to decrease the robustness of regression models, robust training and retraining results in smaller reachable sets and thus ensures that close inputs are mapped

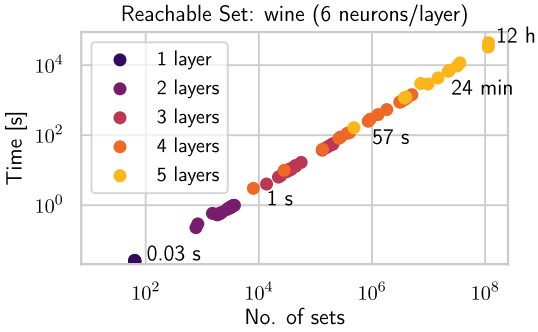


Fig. 10 Run time of RsO vs. worst case no. of subsets required to approximate the reachable set (wine data set, acc. 1,2 layers: 93%, acc. 3-5 layers: 97%)

to close outputs. Thus, robust training and retraining both improve robustness properties without significantly reducing prediction accuracy.

Explainability: reature ranking for classifiers and regression models Reachable sets enable the importance of features to be quantified w.r.t. a model output. To quantify the influence of feature f_1, we define a box-shaped input set with a large perturbation δ on f_1, while the perturbation on the remaining features is small. The size of the reachable set corresponding to $\hat{Z}_{f_1}$ captures the variation in the predictions caused by varying f_1 and thus quantifies the influence of f_1. Since the exact size/volume of $\hat{Z}_{f_1}$ is inefficient to compute (Gover and Krikorian 2010), we approximate it using the interval hull. Here, we use the scaled version of the volume that considers the dimensionality d of the zonotope: $V(\text{IH}(Z)) = \left(2\prod_i \delta g_i\right)^{\frac{1}{d}}$ where $\delta g = \sum_i |g_i|$. The volume of the reachable set is approximated by the sum of all interval hull volumes.Figure 9 illustrates the three most important features for samples of the wine data set computed by our RsO approach in comparison with two other approaches: the integrated gradients method (igm) (Sundararajan et al. 2017) and local interpretable model-agnostic explanations (lime) (Ribeiro et al. 2016). RsO identifies four possibilities for the most important feature: *f*12 (blue, ≈ 30% of samples), *f*9 (teal, ≈ 10% of samples), *f*7 (bright green, ≈ 50% of samples) and *f*6 (yellow, ≈ 20% of samples). Igm identifies three of these possibilities, while lime identifies five possibilities. Overall, the rankings of RsO, igm and lime are of different complexity in terms of different features. The most (second-most/third-most) important feature identified by igm adopts 2-3 possibilities, by lime 5-6 possibilities and by RsO 4-6 possibilities. Consequently, the complexity of the feature ranking computed by RsO is between the one obtained by igm and lime.

Reachable set approximation: analysis of the limits. RsU and RsO approximate the reachable set of a ReLU network layer-by-layer. Within each layer, they compute a linear transformation (defined by the weights and biases) and approximate the outcome of applying ReLU by a set of convex subsets (zonotopes). The number of subsets required to approximate ReLU(Z) is the bottleneck of our approaches. Worst case, applying ReLU on $Z = (c \mid G)$, $c \in \mathbb{R}^D$, $G \in \mathbb{R}^{n \times D}$ results in 2^D subsets/zonotopes. Considering a neural network with K layers of $D_1, D_2, D_3, \ldots D_K$ neurons, the reachable set approximation requires up to $2^{\sum_{k=1}^{K} D_k}$ subsets/zonotopes. Figure 10 illustrates that the run time of RsO linearly increases with the number of subsets and thus exponentially increases with the number of neurons in the worst case. Thus, the number of neurons limits the applicability of our approaches on large neural networks.

To improve this, we propose an extension, which restricts the amplification number and the total number of zonotopes (see Section Balancing approximation tightness and run time), which is applicable to larger neural networks. Results on this extension on

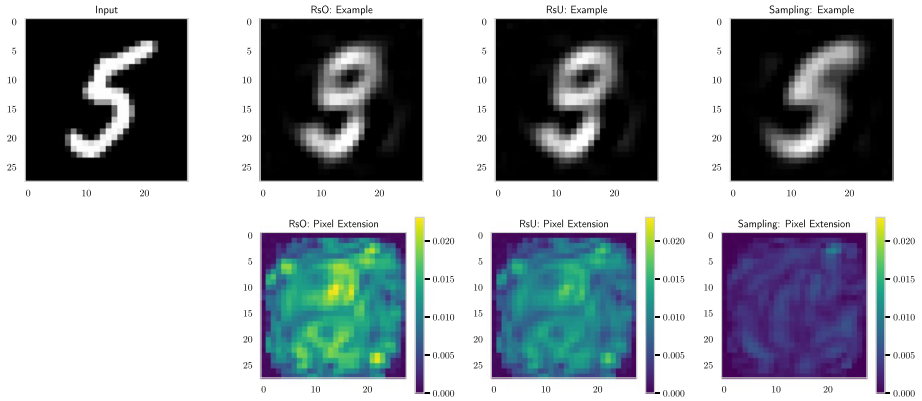


Fig. 11 Analysis of an autoencoder (MNIST, cube, $\epsilon = 0.001$). First row: input image, output image drawn from the reachable set approximated by RsO (second), RsU (third) and sampling (fourth). Second row: extension/size of the pixel range corresponding to the reachable sets computed by RsO (left), RsU (middle) and sampling (right). The smaller the ranges computed by RsO and the larger the ranges computed by RsU or sampling the better is the performance

robustness verification and for the analysis of autoencoders are presented in the next section and in the appendix (see Sect. 6.2).

Autoencoder analysis To illustrate the strength of our approach, we compare reachable sets obtained by RsO and RsU with a sampling-based set approximation. We approximate the reachable set of an autoencoder (three hidden layers, $60 \times 30 \times 60$ neurons) with respect to a cube shaped input set with $\epsilon = 0.001$. RsO and RsU are restricted such that the maximum amplification of a zonotope is $A = 100$ and the overall number of zonotopes is less or equal to $B = 1000$. To compare with RsO and RsU we introduce a simple baseline based on sampling. This sampling approach chooses 10^9 points among the vertices of the cube shaped input set and computes the corresponding outputs. The set spanned by these 10^9 outputs is used to approximate the exact reachable set.

Since we consider autoencoder models, the reachable set consists of pictures from the same space as the input. To visualize the properties of the reachable sets computed by RsO, by RsU and by the sampling approach, we draw example pictures from the reachable sets. Furthermore, we compute the extension/size of the range of each pixel based on the reachable set under consideration (Fig. 11).

Even though we restrict the number of convex subsets to 1000, RsO and RsU result in similar example pictures and similar extensions for each pixel (see Fig. 11, second row and third row). This illustrates that our approximations are tight and close to the exact reachable set, *since the exact reachable set is enclosed by the under- and over-approximation.* In comparison to RsU, the sampling approach results in pixel extensions that are about two times smaller/worse and example pictures that are too close to the image reconstructed from the original input. Thus, sampling 10^9 instances from the input set and computing the corresponding outputs still leads to a dramatic underestimation of the exact reachable set. This shows that RsU outperforms the sampling approach, even if we restrict the overall number of zonotopes and the possible amplification. In conclusion, these results highlight the fact that computing an upper bound (RsO) and a lower bound (RsU) to the reachable set of neural networks provides more information on the mapping of networks than sampling.

5 Conclusion

We propose RsO and RsU as two efficient approaches for over- and under-approximating the reachable sets of ReLU networks. Approximated reachable sets are applicable to the analysis of neural network properties: we analyze and enhance the (non-)robustness properties of both classifiers and regression models. Our approach outperforms PGD attacks as well as state-of-the-art methods of verification for classifiers with respect to non-norm bound perturbations. Reachable sets provide more information than a binary robustness certificate. We use this information for class-specific verification, robustness quantification, robust training, distinguishing between reliable and non-reliable predictions, ranking features according to their influence on a prediction and analyze autoencoders.

Appendix

Pseudocode

Algorithm 1 and 2 show how we under-/over-approximate the outcome of applying ReLU on a zonotope, while Algorithm 3 and 4 show how the reachable set of a neural network is approximated with and without limitations on the number of used subsets.

Algorithm 1: RsO over-approximates applying ReLU on a zonotope

Input: Zonotope $Z = (c \mid G)$, Maximum number MaxAmp of subsets used to approximate one zonotope
Output: Set of zonotopes $RS = \{\hat{Z}_n\}_n$ that over-approximates ReLU(Z)

Compute index sets R_n, R (see Equation 3);
Project Z: $\forall i, \forall d \in R_n : c_d = 0$ and $G[i, d] = 0$;
Compute quadrants with $S_k \subseteq Z$: $\{R_k\}_k = \mathcal{P}(R)$ = power set of R;
Initialize $RS = \{\}$;
if $|\mathcal{P}(R)| > MaxAmp$ **then**
    Compute interval hull $\mathrm{IH}(Z) := \left[c - \sum_i |g_i|, c + \sum_i |g_i|\right]$;
    Restrict IH to its positive parts ;
    $Z_{\mathrm{IH}} = (c|G_{\mathrm{IH}})$ with $G_{\mathrm{IH}} = \mathrm{diag}(\sum_i |g_i|)$;
    $RS = \{\mathrm{Z}_{\mathrm{IH}}\}$;
else
    for $R_k \in \mathcal{P}(R)$ **do**
        Overapproximate S_k by $\hat{Z}_k$ (see Equation 4);
        $RS = RS \cup \{\hat{Z}_k\}$
return RS;

Algorithm 2: RsU under-approximates applying ReLU on a zonotope

Input: Zonotope $Z = (c \mid G)$, Maximum number MaxAmp of subsets used to approximate one zonotope
Output: Set of zonotopes $RS = \{\hat{Z}_n\}_n$ that under-approximates ReLU(Z)

Compute index sets R_n, R (see Equation 3);
Project Z: $\forall i, \forall d \in R_n : c_d = 0$ and $G[i, d] = 0$;
Compute quadrants with $S_k \subseteq Z$: $\{R_k\}_k = \mathcal{P}(R)$ = power set of R;
Initialize $RS = \{\}$;
for $R_k \in \mathcal{P}(R)$ **do**
    Underapproximate S_k by $\hat{Z}_k$ (see Equation 15);
    $RS = RS \cup \{\hat{Z}_k\}$;
    if $|RS| > MaxAmp$ **then**
        break ;
return RS;

Algorithm 3: PropZ propagates zonotope through ReLU network

Input: Zonotope $Z^0 = (c^0 \mid G^0)$, approximation method (RsO or RsU)
Output: Set of zonotopes $RS = \{Z_n\}_n$ that approximates the reachable set

Initialize set of zonotopes $RS = \{Z^0\}_n$
for $k \leftarrow 1$ ***to*** K `// iterate over layers`
do
    $RS' = \{\}$;
    $RS'' = \{\}$;
    for $Z \in RS$ **do**
        Linear transformation: $Z' = \text{lintrans}(Z)$ (see Equation 2) ;
        $RS' = RS' \cup \{Z'\}$;
    if *over approximate* **then**
        for $Z \in RS'$ **do**
            Apply ReLU activation function: $RS'' = RS'' \cup \text{RsO}(Z, \infty)$;
    if *under approximate* **then**
        for $Z \in RS'$ **do**
            Apply ReLU activation function: $RS'' = RS'' \cup \text{RsU}(Z, \infty)$;
    $RS = RS''$;
return RS;

Algorithm 4: PROPZLIMIT propagates zonotope through ReLU network (limited no. subsets)

Input: Zonotope $Z^0 = (c^0 \mid G^0)$, approximation method (RsO or RsU), Maximum number of zonotopes MaxZono, Maximum amplification MaxAmp
Output: Set of zonotopes $RS = \{Z_n\}_n$ that approximates the reachable set

```
Initialize set of zonotopes RS = {Z^0}_n
for k ← 1 to K // iterate over layers
do
    RS' = {} ;
    RS'' = {} for Z ∈ RS do
        Linear transformation: Z' = lintrans(Z) (see Equation 2) ;
        RS' = RS' ∪ {Z'} ;
    if over approximate then
        for Z ∈ RS' do
            Apply ReLU activation function: RS'' = RS'' ∪ RsO(Z, MaxAmp) ;
    if under approximate then
        for Z ∈ RS' do
            Apply ReLU activation function: RS'' = RS'' ∪ RsU(Z, MaxAmp) ;
    RS = RS'' ;
    if |RS| ≥ MaxZono then
        Finde smallest zonotopes ∈ RS ;
        Remove smallest zonotopes from RS ;
        if over approximate then
            Union smallest zonotopes over approximativly by interval hull;
            Add union to RS;
return RS;
```

Extension of RsO and RsU for large(r) neural networks

The subsection "Reachable Set Approximation: Analysis of the Limit" (page 17) shows that the number of zonotopes required to approximate the reachable set might increase exponentially with the number of neurons of the neural network in the worst case. Thus, we propose an extension (see page 9: "Balancing approximation tightness and run-time") that allows to restrict the number of total subsets (max. zono.) and the amplification (max. amp.). The maximum amplification is the maximum number of subsets used to approximate ReLU(Z) w.r.t. the zonotope Z subjected to ReLU, while the maximum number of zonotopes is the maximum number of zonotopes w.r.t. to the whole neural network that is used to approximate the reachable set. This restriction allows to use RsO for robustness verification of larger neural networks. To illustrate how these restrictions affect the tightness of our approximations, we compare the performance of RsO for different max. zono. and max. amp. values on a neural network with 5 hidden layers of 30 neurons on the FashionMNIST data set (see Fig. 12). Without limitations, RsO might require up to $2^{150} \approx 1.43 \cdot 10^{45}$ subsets.

Figure 12 illustrates that the number of robustness certificates increases with the maximum amplification (left plot). Furthermore, the number of robustness certificates increases with max. zono. up to 10, 000, but choosing larger max. zono. does not result in a further increase of robustness certificates (right plot). Thus, to obtain tight approximations

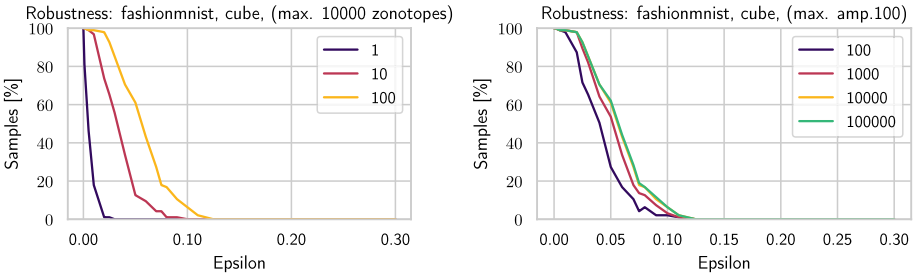


Fig. 12 Performance of RsO on robustness verification using fashionmnist (classification acc. 95%) with different max. amp. and fixed max. zono. (left), with fixed max. amp. and different max. zono. (right)

the max. amp. should be chosen as large as possible and feasible, while the max. zono. should be chosen as small as possible but as large as necessary to obtain the maximum performance.

Acknowledgements This research was supported by BMW AG. We would like to thank Marten Lienen for help with the toolbox that was used to compute the exact reachable set.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A. V., & Criminisi, A. (2016). Measuring neural net robustness with constraints. In *NeurIPS*, Vol. 29.

Brooks, T. F., Pope, D. S., & Marcolini, A. M. (1989). Airfoil self-noise and prediction. *NASA Technical Reports*.

Bunel, R., Turkaslan, I., Torr, P. H., Kohli, P., & Kumar, M. P. (2018). A unified view of piecewise linear neural network verification. In *NeurIPS*, Vol. 31, PP. 4795–4804.

Dua, D., & Graff, C. (2017). UCI machine learning repository. In *University of California*.

Ehlers, R. (2017). Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis*, PP. 269–286.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*.

Forina, M., Leardi, R., Armanino, C., & Lanteri, S. (1990). Parvus: An extendable package of programs for data exploration. *Journal of Chemometrics*.

Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., & Vechev, M. (2018). Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *IEEE Symposium on Security and Privacy*, PP. 3–18.

Goodfellow, I. J., Shlens, J., & Szegedy, C. (2015). Explaining and harnessing adversarial examples. *ICLR*.

Gover, E., & Krikorian, N. (2010). Determinants and the volumes of parallelotopes and zonotopes. *Linear Algebra and its Applications, 433,* 28–40.

Gowal, S., Dvijotham, K., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Arandjelovic, R., Mann, R., & Kohli, P. (2019). Scalable verified training for provably robust image classification. In *ICCV*, PP. 4841–4850.

Harrison, D., & Rubinfeld, D. L. (1978). Hedonic prices and the demand for clean air. *Journal of Environmental Economics and Management, 5,* 81–102.

Hein, M., & Andriushchenko, M. (2017). Formal guarantees on the robustness of a classifier against adversarial manipulation. In *NeurIPS*, Vol. 30.

Jossinet, J. (1996). Variability of impedivity in normal and pathological breast tissue. *Medical and Biological Engineering and Computing, 34,* 346–350.

Katz, G., Barrett, C. W., Dill, D. L., Julian, K., & Kochenderfer, M. J. (2017). Reluplex: An efficient SMT solver for verifying deep neural networks. *CAV, 10426,* 97–117.

Kühn, W. (1998). Rigorously computed orbits of dynamical systems without the wrapping effect. *Computing, 61,* 47–67.

LeCun, Y., Cortes, C., & Burges, C. J. (2010). *Mnist handwritten digit database*. NYU: Courant Institute.

Liu, C., Arnon, T., Lazarus, C., Barrett, C. W., & Kochenderfer, M. J. (2019). Algorithms for verifying deep neural networks. *Foundations and Trends in Optimization, 4,* 244–404.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2018). Towards deep learning models resistant to adversarial attacks. *ICLR*.

Mirman, M., Gehr, T., & Vechev, M. (2018). Differential abstract interpretation for provably robust neural networks. *ICML, 80,* 3578–3586.

Nash, W. J., Sellers, T. L., Talbot, S. R., Cawthorn, A. J., & Ford, W. B. (1994). The population biology of abalone (haliotis species) in tasmania. i. Blacklip abalone (h. rubra) from the north coast and islands of bass strait. *Sea Fisheries Division, Technical Report*, 48.

Raghunathan, A., Steinhardt, J., & Liang, P. (2018). Semidefinite relaxations for certifying robustness to adversarial examples. In *NeurIPS*, Vol. 31.

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "why should i trust you?": Explaining the predictions of any classifier. In *SIGKDD*, PP. 1135–1144.

Ruan, W., Huang, X., & Kwiatkowska, M. (2018). Reachability analysis of deep neural networks with provable guarantees. In *IJCAI*, PP. 2651–2659.

Singh, G., Ganvir, R., Püschel, M., & Vechev, M. (2019a). Beyond the single neuron convex barrier for neural network certification. *NeurIPS*, 32.

Singh, G., Gehr, T., Mirman, M., Püschel, M., & Vechev, M. (2018). Fast and effective robustness certification. In *NeurIPS*, Vol. 31.

Singh, G., Gehr, T., Püschel, M., & Vechev, M. (2019). An abstract domain for certifying neural networks. In *Proceedings of the ACM Programming Languages*, 3.

Singh, G., Gehr, T., Püschel, M., & Vechev, M. (2019). Boosting robustness certification of neural networks. *ICLR*.

Steinhardt, J., Koh, P. W., & Liang, P. (2017). Certified defenses for data poisoning attacks. In *NeurIPS*, Vol. 30, PP. 3520–3532.

Street, N., Wolberg, W., & Mangasarian, O. L. (1999). Nuclear feature extraction for breast tumor diagnosis. *Biomedical Image Processing and Biomedical Visualization, 1905,* 861–870.

Sundararajan, M., Taly, A., & Yan, Q. (2017). Axiomatic attribution for deep networks. *ICML*, 79.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2014). Intriguing properties of neural networks. *ICLR*.

Tjeng, V., Xiao, K. Y., & Tedrake, R. (2019). Evaluating robustness of neural networks with mixed integer programming. *ICLR*.

Wong, E., & Kolter, J. Z. (2018). Provable defenses against adversarial examples via the convex outer adversarial polytope. *ICML, 80,* 5283–5292.

Xiang, W., Tran, H.-D., & Johnson, T. (2017). Reachable set computation and safety verification for neural networks with relu activations. *CoRR*.

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms. *Zalando SE*.

Zhang, H., Cisse, M., Dauphin, Y. N., & Lopez-Paz, D. (2018). Mixup: Beyond empirical risk minimization. *ICLR*.