Reachable Sets of Classifiers & Regression Models: (Non-)Robustness Analysis and Robust Training

Anna-Kathrin Kopetzki
Department of Informatics
Technical University of Munich
Munich, Germany
kopetzki@in.tum.de

Stephan Günnemann
Department of Informatics
Technical University of Munich
Munich, Germany
guennemann@in.tum.de

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 analyze and enhance the robustness properties of both classifiers and regression models. This is in contrast to existing works, which only handle 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. We also provide a technique of distinguishing between reliable and non-reliable predictions for unlabeled inputs, quantify the influence of each feature on a prediction, and compute a feature ranking.

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.

Methods that compute an exact reachable set [29] are not feasible, even for tiny neural networks [15]. In this study, we aim to approximate the reachable set such that it can be computed for neural networks used on standard data sets such as MNIST. 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, the output of the ReLU function is under-approximated or over-approximated for each layer. The resulting sets are used to analyze and enhance neural network properties (see Section 4).

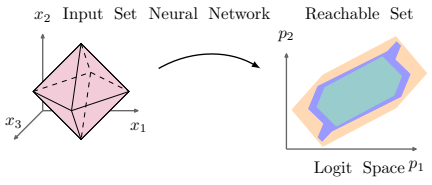


Figure 1: Reachable set over-/under-approximation.

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:

Preprint. Under review.

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 [29] is not applicable even with tiny networks, as shown in [15]. Some techniques that approximate reachable sets, such as [19], 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 [5]. Further developments of this technique include bounds [20, 21, 22]. In addition, set-based approaches are used for robust training [17, 8]. Our work goes significantly beyond the existing approaches. First, our over-approximations are (by construction) subsets of the ones computed in [5] and thus tighter. Second, in addition to over-approximations, we provide an approach for under-approximating the reachable set. Third, in comparison to the improvements presented in [20, 21, 22], our approaches do not require bounds on the layer input.

(Non-)Robustness Verification. Reachable sets are applicable to (non-)robustness verification (see Section 3). Other robustness 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 [28, 18, 1, 12, 23] or L_2-norm [10]. 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 like different brightness of pictures to be analyzed.

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) [16]. In contrast to adversarial attacks, our RsU approach aims to find an entire set of predictions corresponding to an input set.

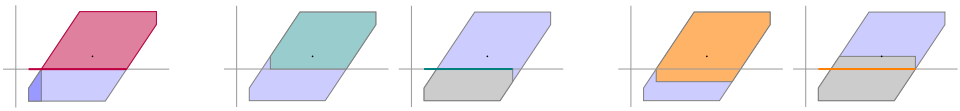
It should be noted that, all the above principles are designed for classification tasks. In contrast, our approach is naturally convenient for regression as well. To further highlight the versatility of our method, we show how to apply it to explaining predictions, including the aspect of reliability.

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 [12]. 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 Figure 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 bias. This linear transformation is computed *exactly and efficiently* for the 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 norm constraints).



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

Figure 2: Application of ReLU to a zonotope (blue) and approximation the outcome.

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:

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

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: $\hat{Z}^0 \overset{W^1, b^1}{\rightarrow} Z^1 \overset{\mathrm{ReLU}}{\rightarrow} \hat{Z}^1 \overset{W^2, b^2}{\rightarrow} Z^2 \overset{\mathrm{ReLU}}{\rightarrow} \dots \overset{W^L, b^L}{\rightarrow} Z^L$, 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 $\mathrm{ReLU}(Z^l)$ is challenging, as it may be a non-convex set, as illustrated in Figure 2a. It is inefficient to further propagate the non-convex set $\mathrm{ReLU}(Z^l)$ through the neural network. Therefore, our core idea is to approximate $\mathrm{ReLU}(Z^l)$ 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 $\mathrm{ReLU}(Z^l)$ 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.

Approximation of ReLU(Z). In the following, we describe how to approximate $\mathrm{ReLU}(Z)$ 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 Figure 2a, 2^{nd} dimension). We consider the three cases separately to approximate each maximum convex subset of $\mathrm{ReLU}(Z)$ by one zonotope. The three cases are distinguished by computing an index set for each case:

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

These index sets can be efficiently obtained through the interval hull of Z [13], where $|.|$ is the element-wise absolute value: $\mathrm{IH}(Z) := [c - \delta g, c + \delta g]$ 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 $\mathrm{ReLU}(Z) = \mathrm{ReLU}(Z')$.*

$$\textit{Here } \mathrm{Proj}_M(Z) = Z' = (c' \mid G') \textit{ where } c'_d = \begin{cases} 0 \text{ if } d \in M \\ c_d \text{ else} \end{cases}, \; g'_{i,d} = \begin{cases} 0 \text{ if } d \in M \\ g_{i,d} \text{ else} \end{cases} \tag{4}$$

All proofs are given in the Appendix. 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. An example of the computation of R, R_k, S_k is given in the Appendix.

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 an extension that restricts the maximum number of zontopes in the Appendix. 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 Figure 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 Figure 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 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 $\alpha_{j,d}$.

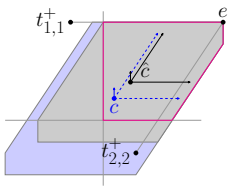


Figure 3: Approximation of $S_0 \subseteq Z$ by $\hat{Z}$.

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:

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

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

$$\forall d \notin R_k : t^+_{j,d} = c_d - 2\,|g_{j,d}| + \sum_i |g_{i,d}|\,, \;\; \text{and} \;\; \forall d \in R_k : t^-_{j,d} = c_d + 2\,|g_{j,d}| - \sum_i |g_{i,d}|$$

$$\forall d \notin R_k : \alpha_{j,d} = 1 - \frac{|t^+_{j,d}|}{2|g_{j,d}|} \text{ if } t^+_{j,d} < 0,\ 1 \text{ else}, \;\; \forall d \in R_k : \alpha_{j,d} = 1 - \frac{|t^-_{j,d}|}{2|g_{j,d}|} \text{ if } t^-_{j,d} > 0,\ 1 \text{ else} \tag{5}$$

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):

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 5. Then $S_k \subseteq \hat{Z}_k$.*

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: $\mathrm{ReLU}(S_k) = \mathrm{Proj}_{R_k}(S_k) \subseteq \mathrm{Proj}_{R_k}(\hat{Z}_k)$.

Under-approximation of ReLU(Z). Under-approximating 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 $\hat{Z}_k \subseteq S_k$:

$$\hat{Z}_k = \arg\max_{\hat{Z}} V(\hat{Z}) \text{ subject to } \hat{Z} \subseteq S_k \tag{6}$$

How can we instantiate Equation 6 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 $d \notin R_k$: $\hat{c}_d - \sum_i |\hat{g}_{i,d}| \geq 0 \ \ \forall d \notin R_k$ and one that forces the upper bound of the interval hull of $\hat{Z}$ to be negative if $d \in R_k$: $\hat{c}_d + \sum_i |\hat{g}_{i,d}| \leq 0 \ \ \forall d \in R_k$. 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:

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

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

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 $\mathrm{Proj}_{R_k}(\hat{Z}_k)$ and obtain an under-approximation of $\mathrm{ReLU}(S_k)$. See Figure 2b for illustration.

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. 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 such as MNIST. A detailed description of the datasets, network architectures, experimental setups and additional experimental results are provided in the Appendix.

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:

$$\mathrm{s}_b = \min_{p \in R_S} (p_a - p_b) = \min_{Z=(c|G) \in R_S} \left(c_a - c_b - \sum_i \left| g_i^a - g_i^b \right| \right) \tag{7}$$

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 $g_i^a - g_i^b$.

[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.

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 7). 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, WK [28], dz [20], dp [21], dr [22] and es [29], which computes the exact reachable set (implementation [15]). RsU is compared with the success rate of FGSM attacks [6, 25] and PGD attacks [16]. 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. Figure 4 illustrates (non-)robustness verification on the cancer data set (2 hidden layers), MNIST (1 hidden layer) and iris data set (5 hidden layers) for cube-, box- and free-shaped input zonotopes. For robustness verification, we measure

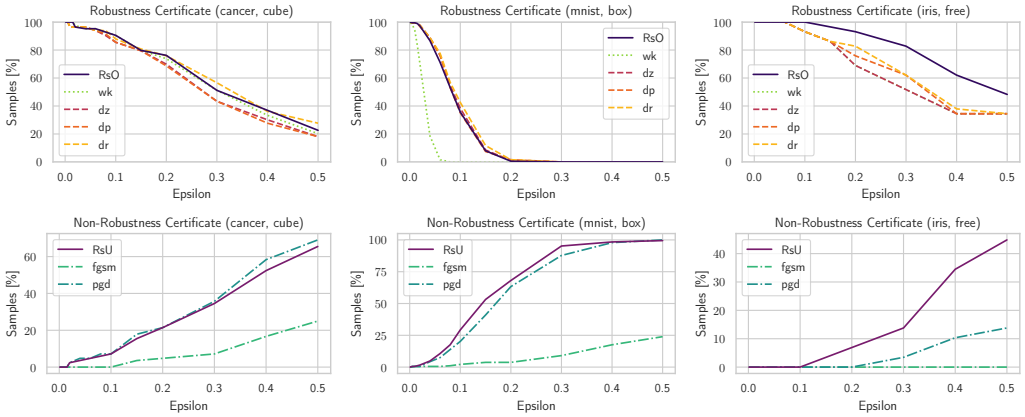


Figure 4: Comparison of (non-)robustness verification on MNIST.

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. A detailed comparison between the cube, box and free settings, run-times and further experimental results on other data sets such as FashionMNIST are given in the Appendix. Since es [29], which computes the exact reachable set, requires too much time even with the smallest neural network architecture (see Appendix), we could not include it in our comparison.

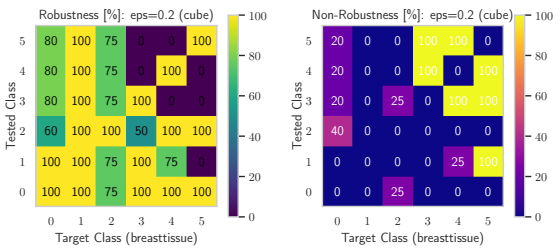


Figure 5: Class specific verification.

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 (3 hidden layers). The authors are of the opinion that distinguishing between the class 3, 4 or 5 is of minor importance, while it is crucial to distinguish these classes from class 1. This is illustrated in Figure 5, 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). Class-specific analysis allows classifiers to be evaluated more specifically and focus on crucial robustness properties.

Classification: Reliability of Predictions. How can we distinguish between reliable and non-reliable predictions if the label is not known, e.g. during test time? We propose an approach called θ-robustness, which does not require label information and is inspired by the following observation: while a wrong prediction (w.r.t. ground truth) can theoretically have a high robustness score, we have noticed that robustness scores corresponding to wrongly predicted inputs are mostly negative or close to zero. Thus, we consider inputs as reliable if their scores (w.r.t. the predicted class) are larger than a threshold θ; they are θ-robust. To define a good threshold θ we first compute robustness scores for the whole data set (including wrongly predicted points). Then we compute the true-above-rate (TAR) and the false-above-rate (FAR) for different thresholds θ: $\mathrm{TAR}(\theta) = \frac{\text{No. corr. pred., } \theta\text{−rob. points}}{\text{No. corr. pred. points}}$, $\mathrm{FAR}(\theta) = \frac{\text{No. wrong. pred., } \theta\text{−rob. points}}{\text{No. wrong. pred. points}}$. These rates are inspired by true-positive/false-positive rates used in receiver operating characteristic curves (ROC). The threshold θ is chosen such that it maximizes TAR and minimizes FAR. Figure 6 illustrates several tested thresholds on Fashion-MNIST (1 hidden layer). According to θ-robustness verification with $\theta = 2.0$ of the test set, $87.0\ \%$ of correctly classified samples are identified as reliable while $87.5\ \%$ of wrongly classified samples are identified as non-reliable. Thus, θ-robustness makes it possible to differentiate between reliable and non-reliable predictions if the label of an input is unknown.

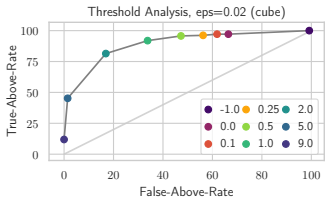


Figure 6: θ-robustness

Classification: Robust Training. The robustness scores as defined in Equation 7 are directly used in robust training by incorporating them into the loss function, e.g. as follows:

$$L_{\text{rob}} = L_{\text{pred}} + \mathbb{I}[\text{pred} = \text{target}] \cdot \max_b \text{ReLU}\left(-s_b\right) \tag{8}$$

where L_{pred} is the cross-entropy loss and $\mathbb{I}[\text{pred} = \text{target}] = 1$ 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 shows that robust training (L_{rob}) or retraining (warm-start with a normally trained model, further training with L_{rob}) against box-shaped perturbations ($\varepsilon = 0.05$, FashionMNIST, 1 hidden layer) enhances the robustness of the neural network. The accuracy of the models and the mean training time of an epoch are as follows: normal $91\ \%$, $0.0007s$, retrain $91\ \%$, $0.16s$, robust $89\ \%$, $0.22s$. Thus, robust training enhances robustness w.r.t. cube-, box- and free-shaped perturbations without decreasing accuracy.

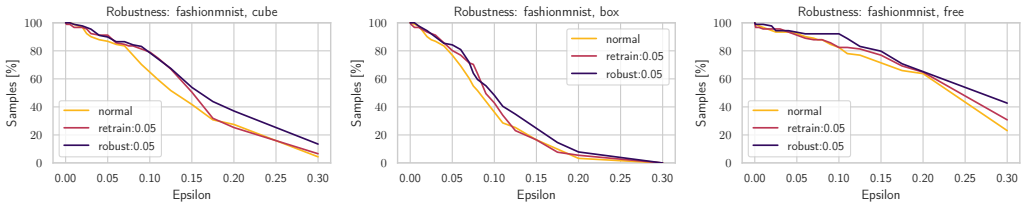


Figure 7: Evaluation of robust training against box-shaped inputs on FashionMNIST.

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 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: $l_a = |\max_{u \in R_S} u_a - \min_{v \in R_S} v_a|$. 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):

$$L_{\text{rob}} = L_{\text{val}} + \text{ReLU}\left(\max_a l_a - l_{\text{in}}\right) \tag{9}$$

If l_a is larger than l_{in} the second term of L_{rob} is positive, otherwise it is zero. We compare three different training modes: normal (training with Huber loss), retrain (warm-start with a normally trained model, and further training with L_{rob}) and robust (training with L_{rob}). Figure 8 illustrates the training and robustness analysis of the three regression models obtained, based on the abalone data set (2 hidden layers). Robust training (retrain or robust) results in smaller reachable sets and

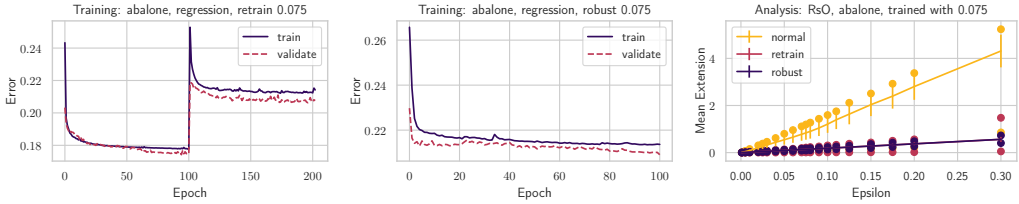


Figure 8: Robust training and analysis of regression models.

thus ensures that close inputs are mapped to close outputs. The mean Huber loss error on the abalone test set are: normal 0.20, retrain 0.24 and robust 0.24. Thus, robust training improves robustness properties without significantly reducing prediction accuracy. A discussion of the run-time of the robustness analysis, robust training and further experimental results is given in the Appendix.

This robustness definition, analysis and robust training proposed for regression models can be applied to any network. For illustration a case study on an autoencoder is provided in the Appendix.

Explainability: Feature Ranking for Classifiers & 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 [7], we approximate it using the interval hull. Here, we use the scaled version of the volume that considers the dimensionality d of the zonotope: $V\left(\text{IH}(Z)\right) = (2\prod_i \delta g_i)^{\frac{1}{d}}$ 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 feature ranking for the classes 0 and 1 of the wine data set (13 features, 1 hidden layer). As demonstrated, our model enables features to be ranked w.r.t. the considered class while being stable w.r.t. the input size δ of the zonotope.

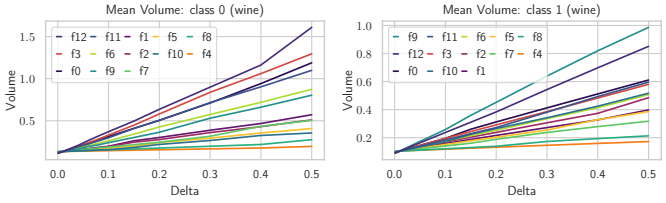


Figure 9: Feature ranking.

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, and ranking features according to their influence on a prediction.

6 Acknowledgments

We would like to thank Marten Lienen for help with the toolbox that was used to compute the exact reachable set [15]. Furthermore, we thank the UCI Machine Learning Repository [2] for providing data sets.

References

[1] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. V. Nori, and A. Criminisi. Measuring neural net robustness with constraints. *CoRR*, 2016.

[2] D. Dua and C. Graff. UCI machine learning repository, 2017.

[3] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Ann. Eugen.*, 1936.

[4] M. Forina, R. Leardi, C. Armanino, and S. Lanteri. *PARVUS: An Extendable Package of Programs for Data Exploration*. 1998.

[5] 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 Symp. Sec. & Privacy*, 2018.

[6] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *Int. Con. Learn. Rep.*, 2015.

[7] E. Gover and N. Krikorian. Determinants and the volumes of parallelotopes and zonotopes. *Lin. Alg. Appl.*, 2010.

[8] S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelovic, T. Mann, and P. Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *CoRR*, 2018.

[9] Rubinfeld D.L. Harrison, D. Hedonic prices and the demand for clean air. *J. Environ. Economics and Management*, 1978.

[10] M. Hein and M. Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *Con. Neur. Inform. Process. Sys.*, 2017.

[11] J. Jossinet. Variability of impedivity in normal and pathological breast tissue. *Med. & Biol. Eng. & Comp.*, 1996.

[12] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. *CoRR*, 2017.

[13] W. Kühn. Rigorously computed orbits of dynamical systems without the wrapping effect. *Comp.*, 1998.

[14] Y. LeCun, C. Cortes, and C.J. Burges. Mnist handwritten digit database. *ATT Labs [Online]*, 2010.

[15] C. Liu, T. Arnon, C. Lazarus, C. W. Barrett, and M. J. Kochenderfer. Algorithms for verifying deep neural networks. *CoRR*, 2019.

[16] A. Madry, A.Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *Int. Con. Learn. Rep.*, 2018.

[17] M. Mirman, T. Gehr, and M. Vechev. Differential abstract interpretation for provably robust neural networks. In *Int. Con. Mach. Learn.*, 2018.

[18] A. Raghunathan, J. Steinhardt, and P. Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *Con. Neur. Inform. Process. Sys.*, 2018.

[19] W. Ruan, X. Huang, and M. Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. *Int. Joint Con. Art. Intel.*, 2018.

[20] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev. Fast and effective robustness certification. In *Adv. in Neur. Inform. Process. Sys.* 2018.

[21] G Singh, T. Gehr, M. Püschel, and M. Vechev. An abstract domain for certifying neural networks. *Proc. ACM Prog. Lang.*, 2019.

[22] G. Singh, T. Gehr, M. Püschel, and M. Vechev. Boosting robustness certification of neural networks. In *Int. Con. Learn. Rep.*, 2019.

[23] J. Steinhardt, P. W. Koh, and P. Liang. Certified defenses for data poisoning attacks. *CoRR*, 2017.

[24] N. Street, W. Wolberg, and Mangasarian. Nuclear feature extraction for breast tumor diagnosis. *Proc., Soc. Photo-Opt. Instrum. Eng.*, 1999.

[25] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *Int. Con. Learn. Rep.*, 2014.

[26] A. M. Marcolini T. F. Brooks, D. S. Pope. Airfoil self-noise and prediction. *Technical report, NASA RP-1218*, 1989.

[27] S. R. Talbot A. J. Cawthorn W. B. Ford W. J. Nash, T. L. Sellers. 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*, 1994.

[28] E. Wong and J. Z. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. *Int. Con. Learn. Rep.*, 2017.

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

[30] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, 2017.

7 Appendix – Reachable Sets of Classifiers & Regression Models: (Non-)Robustness Analysis and Robust Training

7.1 Proof: Projection of Zonotopes

Recall that projecting a zonotope results in a more compact representation but does not change the output set (Theorem 1):

Theorem. *Let Z be an arbitrary zonotope and $Z' = \mathrm{Proj}_{R_{\mathrm{neg}}}(Z)$, then* $\mathrm{ReLU}(Z) = \mathrm{ReLU}(Z')$.

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

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

It holds that:

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

□

7.2 Proof: Over-approximation of Zonotopes

Remember how we over-approximate a subset S_k (Theorem 2):

Theorem. *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 $\hat{c}$ and generators as defined in Equation 5. 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:

$$\begin{aligned} p &= c + \sum_j \beta_j g_j = \hat{c} - \sum_j s_j(1-\alpha_j) o_j g_j + \sum_j \beta'_j o_j g_j \\ &= \hat{c} + \sum_j (\beta'_j - s_j(1-\alpha_j)) o_j g_j = \hat{c} + \sum_j \frac{\beta'_j - s_j(1-\alpha_j)}{\alpha_j} o_j \hat{g}_j \end{aligned} \tag{10}$$

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 Theorem 4). 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 Theorem 5). 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 $p = \hat{c} + \sum_j \hat{\beta}_j \hat{g}_j \in \hat{Z}_k \Rightarrow \forall p \in S_k : p \in \hat{Z}_k \;\text{ and }\; S_k \subseteq \hat{Z}_k$. □

Theorem 4. *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 5. 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 $t^+_{j,d} < 0$.

$$0 \leq p_d \tag{11}$$

$$\Leftrightarrow t^+_{j,d} + 2\frac{1}{2}\frac{|t^+_{j,d}|}{|g_{j,d}|}|g_{j,d}| \leq c_d + \sum_i \beta_i g_{i,d} \tag{12}$$

$$\Leftrightarrow t^+_{j,d} + 2(1-\alpha_j)|g_{j,d}| \leq c_d + \sum_i \beta_i g_{i,d} \tag{13}$$

$$\Leftrightarrow c_d + \sum_i |g_{i,d}| - 2\,|g_{j,d}| + 2(1-\alpha_j)|g_{j,d}| \leq c_d + \sum_i \beta_i g_{i,d} \tag{14}$$

$$\Leftrightarrow \sum_{i,i\neq j} o_{i,d} g_{i,d} - |g^d_j| + 2(1-\alpha_j)|g_{j,d}| \leq \sum_{i,i\neq j} \beta_i g_{i,d} + \beta_j g_{j,d} \tag{15}$$

$$\Leftrightarrow \sum_{i,i\neq j} (o^d_i - \beta_i) g_{i,d} + (2(1-\alpha_j) - 1)|g_{j,d}| \leq \beta'_j |g_{j,d}| \tag{16}$$

$$\Leftrightarrow \underbrace{\frac{1}{|g_{j,d}|}\sum_{i,i\neq j}(o_{i,d} - \beta_i) g_{i,d}}_{\geq 0} + (2(1-\alpha_j) - 1) \leq \beta'_j \tag{17}$$

$$\Rightarrow 2(1-\alpha_j) - 1 \leq \beta'_j \tag{18}$$

We use that $p_d = c_d + \sum_i \beta_i g_{i,d}$ (12), 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 (13), the definition of $t^+_{j,d}$ (14), $o_{j,d} g_{j,d} = |g_{j,d}|$ (15) and $\beta_j g_{j,d} = \beta'_j |g_{j,d}|$ (16). Inequality $\frac{1}{|g_{j,d}|}\sum_{i,i\neq j}(o_{i,d} - \beta_i) g_{i,d} \geq 0$ (17) 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, $\sum_{i,i\neq j}(o_{i,d} - \beta_i) g_{i,d} \geq 0$. □

Theorem 5. *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 5. Consider the definitions used in Theorem 2. Then $\beta'_j \leq 1 - 2(1-\alpha_j)$ if α_j corresponds to a $d \in R_k$.*

Proof. We use that for a point $p \in Z : p_d \leq 0$ in case $d \in R_k$ and $t^-_{j,d} > 0$.

$$0 \geq p_d \tag{19}$$

$$\Leftrightarrow t^-_{j,d} - 2\frac{1}{2}\frac{|t^-_{j,d}|}{|g_{j,d}|}|g_{j,d}| \geq c_d + \sum_i \beta_i g_{i,d} \tag{20}$$

$$\Leftrightarrow t^-_{j,d} - 2(1-\alpha_j)|g_{j,d}| \geq c_d + \sum_i \beta_i g_{i,d} \tag{21}$$

$$\Leftrightarrow c_d - \sum_i |g_{i,d}| + 2\,|g_{j,d}| - 2(1-\alpha_j)|g_{j,d}| \geq c_d + \sum_i \beta_i g_{i,d} \tag{22}$$

$$\Leftrightarrow -\sum_{i,i\neq j} o_{i,d} g_{i,d} + |g_{j,d}| - 2(1-\alpha_j)|g_{j,d}| \geq \sum_{i,i\neq j} \beta_i g_{i,d} i + \beta_j g_{j,d} \tag{23}$$

$$\Leftrightarrow \sum_{i,i\neq j} (-o_{i,d} - \beta_i) g_{i,d} + (1 - 2(1-\alpha_j))|g_{j,d}| \geq \beta'_j |g_{j,d}| \tag{24}$$

$$\Leftrightarrow \underbrace{\frac{1}{|g_{j,d}|}\sum_{i,i\neq j}(-o_{i,d} - \beta_i) g_{i,d}}_{\leq 0} + (1 - 2(1-\alpha_j)) \geq \beta'_j \tag{25}$$

$$\Rightarrow 1 - 2(1-\alpha_j) \geq \beta'_j \tag{26}$$

We use that $p_d = c_d + \sum_i \beta_i g_{i,d}$ (20), 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 (21), the definition of $t^-_{j,d}$ (22), $o_{j,d} g_{j,d} = |g_{j,d}|$ (23) and $\beta_j g_{j,d} = \beta'_j |g^d_j|$ (24). Inequality

$\frac{1}{|g_{j,d}|}\sum_{i,i\neq j}(-o_{i,d}-\beta_i)g_{i,d} \leq 0$ (25) 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 $\beta_i \in [-1,1] \Rightarrow\ (-o_{i,d}-\beta_i)g_{i,d} \leq 0$ and thus, $\sum_{i,i\neq j}(-o_{i,d}-\beta_i)g_{i,d} \leq 0$. □

7.3 Proof: Under-approximation of Zonotopes

Recall how we under-approximate a subset S_k (Theorem 3):

Theorem. *Let $\hat{Z}_k$ be computed from zonotope Z based on α^*, δ^* (Equation 7). 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:

$$\begin{aligned} p &= \hat{c} + \sum_i \hat{\beta}_i \hat{g}_i = c + \sum_i \delta_i g_i + \sum_i \hat{\beta}_i \alpha_i g_i \\ &= c + \sum_i \gamma_i (1-\alpha_i) g_i + \sum_i \hat{\beta}_i \alpha_i g_i = c + \sum_i \left((1-\alpha_i)\gamma_i + \alpha_i \hat{\beta}_i \right) g_i \end{aligned}$$

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:

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

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:

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

Thus, $\hat{Z}_k \subseteq S_k$. □

7.4 Computation of R, R_k and the corresponding S_k

Recall the approximation of $\mathrm{ReLU}(Z)$ and the three cases illustrated in Section 3 (Approximation of $\mathrm{ReLU}(Z)$). In the following, we provide an example how to compute R, R_k and the corresponding S_k.

Example 1. *Consider zonotope $Z = (c \mid G)$ (Figure 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 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.*

7.5 Extension of RsO and RsU for Large Neural Networks

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 all 2^d quadrants. In this case, RsO and RsU approximate each convex subset S_k located in one quadrant by a separate zonotope Z_k. Each of these zonotopes Z_k is an input of the next layer and in the worst case it amplifies the same way as Z.

To apply reachable set computation on large neural networks, we extend our RsO and RsU approach by restricting the overall number of zonotopes by B and the amplification by A. In this context, amplification refers to the number of zonotopes that are used to approximate the convex subsets S_k corresponding to $\mathrm{ReLU}(Z)$, i.e. to one zonotope Z. The amplification is defined by the number of quadrants that contain points of Z. The number of quadrants q that contain points of Z 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 $\left[l^d_{\mathrm{low}}, l^d_{\mathrm{upp}}\right] = [c_d - \delta g_d, c_d + \delta g_d]$, where $\delta g = \sum_i |g_i|$. Next we count the number R_n of dimensions d where $l^d_{\mathrm{low}} < 0$ and $l^d_{\mathrm{upp}} > 0$. The number of quadrants is $q = 2^{R_n}$.

We distinguish between two cases: over-approximation and under-approximation. In the first case, we over-approximate $\mathrm{ReLU}(Z)$ by computing its interval hull and restricting the intervals to the positive portion if $q > A$. Thus, we over-approximate $\mathrm{ReLU}(Z)$ by one convex set instead of q sets. If the overall number of zonotopes becomes larger than B, we approximate the size of each zonotope, keep the largest $B - 1$ zonotopes and union the others over-approximately. As size measure, we use the extension of the interval hull of $Z = (c \mid G)$:

$$\mathrm{size}\,(Z) = \sum_d \log(\delta g)_d, \quad \delta g = \sum_i |g_i|. \tag{27}$$

The union of a set of zonotopes is over-approximated by the interval hull of each zonotope and by minimizing/maximizing over the lower/upper limits of the interval hull. The resulting interval is then transformed into the G-representation of a zonotope:

$$\begin{aligned} l_{\mathrm{low}} &= \min_{Z_k}(c_k - (\delta g)_k), \quad l_{\mathrm{upp}} = \max_{Z_k}(c_k + (\delta g)_k) \\ Z_{\mathrm{uni}} &= (l_{\mathrm{low}} + g_{\mathrm{ext}} \mid \mathrm{diag}(g_{\mathrm{ext}})), \quad g_{\mathrm{ext}} = 0.5(l_{\mathrm{upp}} - l_{\mathrm{low}}) \end{aligned} \tag{28}$$

where $\mathrm{diag}(g_{\mathrm{ext}})$ is a diagonal matrix.

In the second case, we obtain an under-approximation by dropping zonotopes if $q > A$ or the overall number of zonotopes becomes larger than B. Thus, we consider a subset of the quadrants, keep the largest under-approximations and drop the others.

7.6 Details of the Experimental Setup

To evaluate the performance of our reachable set computations, we apply them to robustness and non-robustness verification and robust training and use them to quantify feature importance and to compute trust indicators for predictions. 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, as defined in the main paper, 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** [3, 2]: 3 classes, 4 features, $1-5$ hidden layers of 4 neurons each, accuracy $\geq 96.7\ \%$. **Wine** [4, 2]: 3 classes (cultivars), 13 features, $1-4$ hidden layers of 6 neurons each, accuracy $\geq 96.7\ \%$. **Tissue** [11, 2]: breast tissue probes, 6 classes, 9 features, $1-3$ hidden layers of 8 neurons each, accuracy $\geq 70.0\ \%$. Breast **cancer** Wisconsin (diagnostic) [24, 2]: 2 classes, 30 features, $1-2$ hidden layers of 10 neurons each, accuracy $\geq 96.5\ \%$. **MNIST** [14]: 28×28 gray-scale images, 10 classes, 1 hidden layer of 15 neurons, accuracy $\geq 94.0\ \%$. **Fashion-MNIST** [30]: 28×28 gray-scale images, 10 classes, 1 hidden layer of 15 neurons, accuracy of $92.0\ \%$. Regression Models: **Abalone** [27]: 8 features, 1 output, $1-3$ hidden layers of 6 neurons each. **Housing** [9]: 13 features, 1 output, 1 hidden layer of 13 neurons. **Airfoil** [26]: 5 features, 1 output, $1-4$ hidden layers of 5 neurons each. Autoencoder: **MNIST** [14]: 28×28 gray scale images, 28×28 output, 3 hidden layers of $30 \times 60 \times 30$ neurons. **Fashion-MNIST** [30]: 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).

7.6.1 Detailed 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: $\hat{G} = [\delta I_d, \varepsilon \vec{1}]$.

For feature rankings, the following setting was 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 $G_{i,i} = \varepsilon\ \forall i \neq 1$.

7.7 Further Experimental Results: Classifiers

Comparison with Exact Reachable Set Computations. It was not possible to conduct a meaningful comparison with method es [29, 15] that computes the exact reachable set of a neural network, since it only ran on the smallest neural network (iris data set, neural network with 1 hidden layer of 4 neurons) for the smallest $\varepsilon \in \{0.001, 0.005, 0.01\}$. When performing the exact method es on a cube-shape input with perturbation $\varepsilon = 0.02$, 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.

Run-times and numbers of verified samples of our approaches RsO, RsU and of the exact method es are shown in Table 1. Note that we used a version of [15] 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 [15, 29].

ε	RsO		RsU		es		
	No. robust	Time [ms]	No. non-robust	Time [ms]	No. robust	No. non-robust	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	-	-	not finished > 3d

Table 1: Comparison of RsO and RsU with the exact reachable set computation (es) [29, 15] on 29 correctly classified samples of the iris data set (neural network with 1 hidden layer of 4 neurons, cube setting).

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.

(Non-)Robustness Verification of Classifiers. Detailed numbers on the verification of a classifier trained on Fashion-MNIST are given in Figure 10. On cube- and box-shaped inputs, RsO and

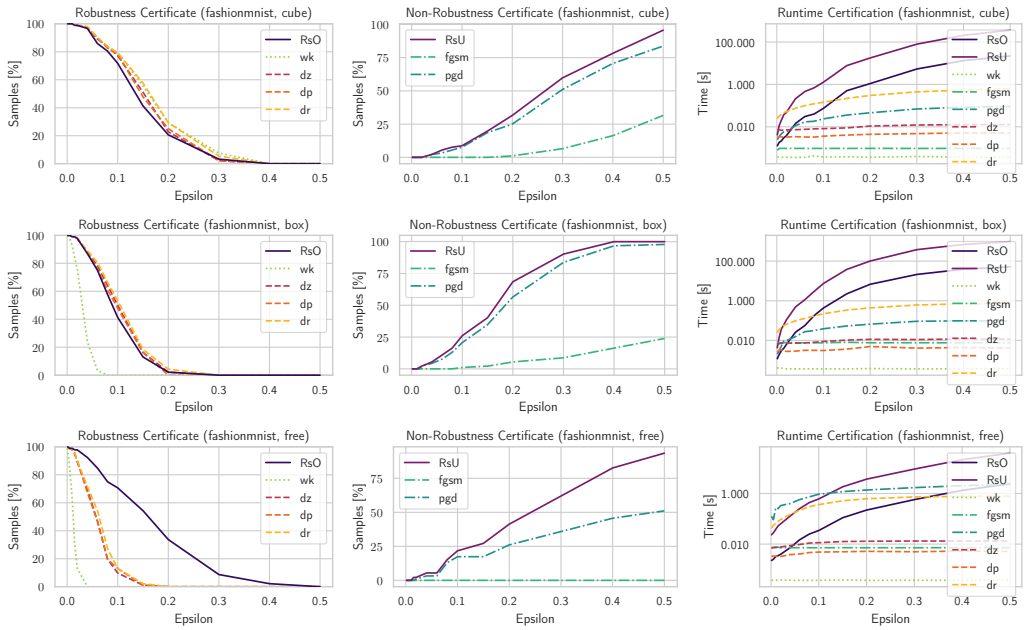


Figure 10: Performance evaluation of RsO and RsU in (non-)robustness verification on Fashion-MNIST using different input sets/perturbations.

RsU perform in a similar way to the state-of-the-art verification techniques dz, dp, dr and PGD

attacks. If we use input sets that allow features to be coupled (free-setting), RsO and RsU outperform dz, dp, dr and PGD attacks. 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.

Class-specific Verification. Class-specific robustness verification allows us to draw conclusions about the concepts a neural network has learned (see Figure 11 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). This indicates that the neural network has extracted the shape and learned its importance for a classification decision.

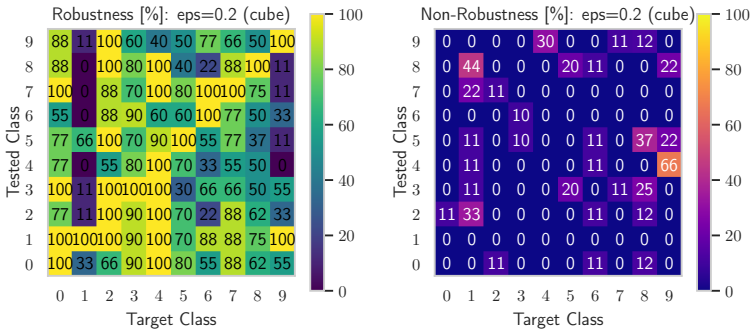


Figure 11: Class specific (non-)robustness verification on the Fashion-MNIST data set.

7.8 Further Experimental Results: Regression Models

In this section we further demonstrate the performance of RsO and RsU in robust training and analysis of regression models (Figure 12).

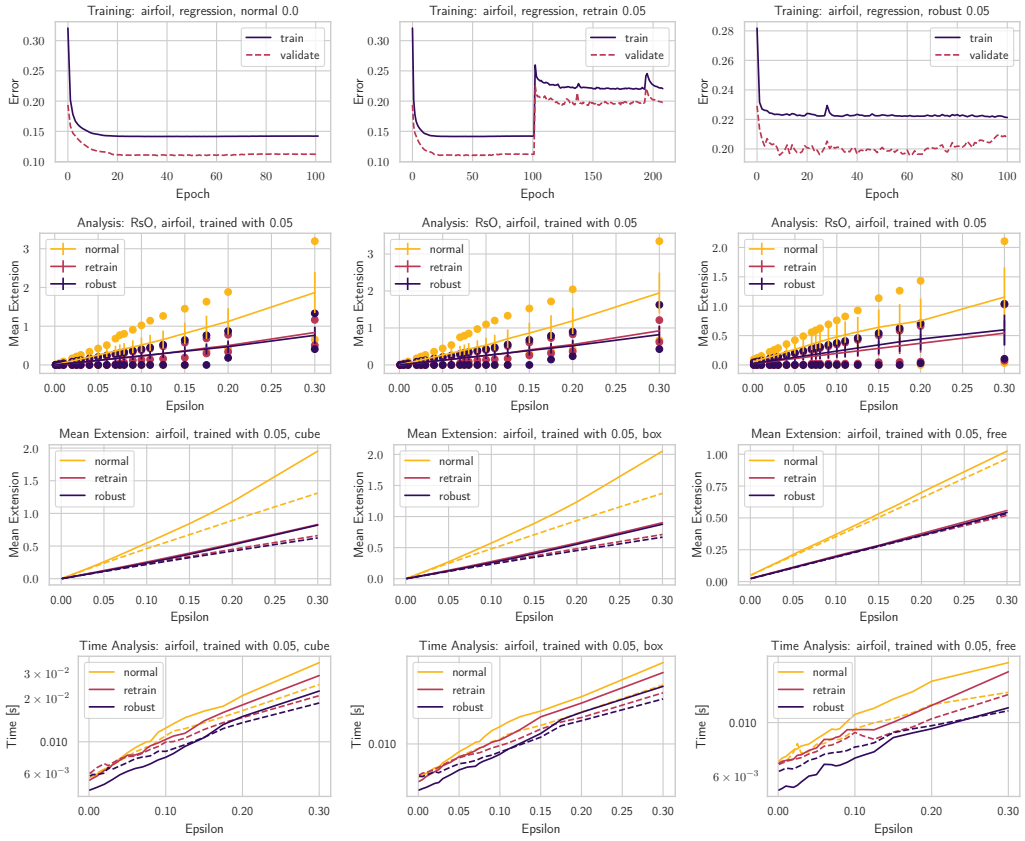


Figure 12: Robust training and robustness analysis on the airfoil data set (RsO: solid lines, RsU: dashed lines).

We use three different training procedures: normal (normally trained neural network), retrain (warm start with normally trained model, further training with the robust loss function) and robust (training with the robust loss function, which is defined in the main section of this paper). Experiments are carried out on a neural network with one hidden layer and we train against cube-shaped perturbations of size $\varepsilon = 0.05$. The mean time of a training epoch is $1.09ms$ for normal training, $5.60ms$ for retraining and $9.47ms$ for robust training.

Our evaluation illustrates two aspects: First, our approximations are tight, since the extensions of the reachable sets computed by RsU (dashed lines) and RsO (solid lines) are close. Second, robust training improves the robustness properties of regression models, since the under-approximated reachable sets of the normal model are larger than the over-approximated reachable sets of the robustly trained models.

7.9 Further Experimental Results: Autoencoder Models

To illustrate the strength of our approach, we compare reachable sets obtained by our extended RsO and RsU approach (see Section 7.5) with a sampling-based set approximation. To do this, 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 $\varepsilon = 0.001$. 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 $B = 1000$. The 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 compare 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 (Figure 13).

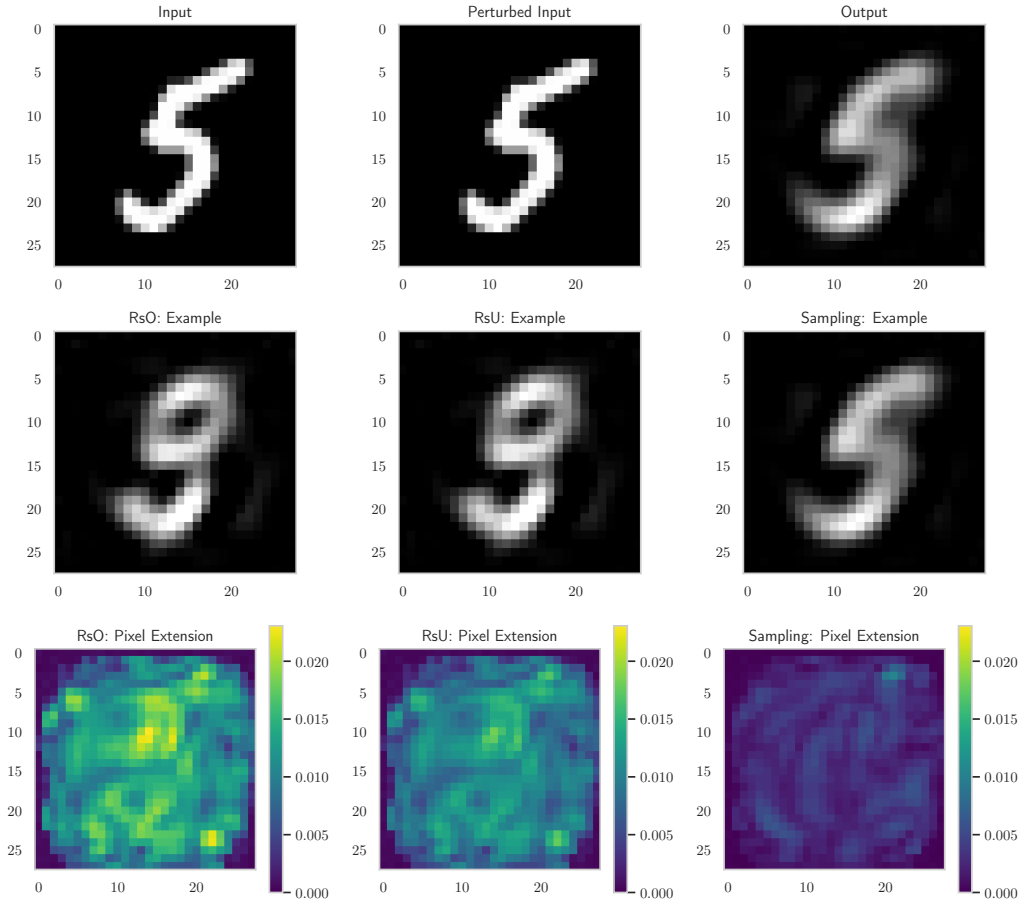


Figure 13: Analysis of an autoencoder (MNIST, cube, $\varepsilon = 0.001$). First row: input image, perturbed input, corresponding output image. Second row: output image drawn from the reachable set approximated by RsO (left), RsU (middle) and sampling (right). Third row: extension/size of the pixel range corresponding to the reachable sets computed by RsO (left), RsU (middle) and sampling (right).

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 Figure 13, second row and third row). This illustrates that our approximations are tight and close to the exact reachable set, as the exact reachable set is enclosed by the under-approximation and the over-approximation. In comparison to RsU, the sampling approach results in pixel extensions that are about two times smaller and example pictures that are close to the image reconstructed from the original input. Thus, sampling

10^9 vertices from the input set and computing the corresponding outputs under-estimates 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.