Differentiable Abstract Interpretation for Provably Robust Neural Networks

Matthew Mirman [1] **Timon Gehr** [1] **Martin Vechev** [1]

Abstract

We introduce a scalable method for training robust neural networks based on abstract interpretation. We present several abstract transformers which balance efficiency with precision and show these can be used to train large neural networks that are certifiably robust to adversarial perturbations.

1. Introduction

Neural networks are increasingly gaining importance in critical areas such as facial recognition and autonomous driving. Szegedy et al. (2013) discovered that even well-trained neural networks can misclassify inputs which are seemingly identical to correctly classified ones. Further, such *adversarial examples* have been constructed physically (Athalye & Sutskever, 2017; Evtimov et al., 2017) as well as in black-box settings (Papernot et al., 2016; 2017).

Gu & Rigazio (2014) provided an early technique for defending against adversarial examples based on adding concrete noise to the training set and removing it statistically. Goodfellow et al. (2014) showed an adversarial attack which generated examples that were misclassified on a wide array of networks, and then demonstrated a defense against this attack, based on explicit training against perturbations generated by the attack. Madry et al. (2018) improved on this style of defense by showing that training against an optimal attack would provide a defense against non-optimal attacks as well. While this technique was highly effective in experiments, Carlini et al. (2017) demonstrated an attack for the safety-critical problem of ground-truthing, where this defense in fact occasionally exacerbated the problem.

As heuristic defenses are insufficient to ensure safety, works such as Gehr et al. (2018) provided methods for certifying local robustness properties of neural networks. Kolter & Wong (2017) demonstrated the first defense against adversarial attacks which provides certificates proving that none of the training examples could be adversarially permuted, as well as bounds on the capability of an adversary to influence performance on a test set. This method is based on computing an overapproximation to the *adversarial polytope*, which describes the set of possible neural network outputs given the region of possible inputs. However, this approach incurs significant accuracy and scalability overheads. Recent work by Raghunathan et al. (2018) provides certifiable robustness, but only for neural networks consisting of two layers. Thus, developing techniques to train large neural networks that can be automatically certified free of robustness violations remains a fundamental challenge.

Abstract Interpretation for Training Neural Networks We address the above challenge by leveraging the classic framework of *abstract interpretation* (Cousot & Cousot, 1977), a general theory for approximating a potentially infinite set of behaviors with a finite representation. This theory has been widely used over the last 40 years to build large-scale automatic code analyzers (Blanchet et al., 2003). We show how to bridge abstract interpretation and gradient-based optimization and how to apply these concepts to train larger networks. Concretely, we compute an approximation to the adversarial polytope and use this approximation as part of our loss function, effectively training the network on entire regions of the input space at once. This *abstract loss* has the advantage of being optimizable via standard techniques such as gradient descent and, as we demonstrate, networks trained in this manner are more provably robust.

Main Contributions Our main contributions are:

- A new method for training neural networks based on abstract interpretation (Sections 2 and 5).
- Novel abstract transformers for the zonotope domain which are parallelizable and suited for differentiation and gradient descent (Sections 3 and 4).
- A complete implementation of the method in a system called DIFFAI[1] together with an extensive evaluation on a range of datasets and architectures. Our results show that DIFFAI improves provability of robustness and scales to large networks (Section 6).

[1]Department of Computer Science, ETH Zurich, Switzerland. Correspondence to: Matthew Mirman <matthew.mirman@inf.ethz.ch>, Martin Vechev <martin.vechev@inf.ethz.ch>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

[1]Available at: `http://diffai.ethz.ch`

2. Robustness and Sound Approximations

In this section we review and formally define the concept of robustness and discuss an approach to robustness via sound, computable approximations.

Let $N_\theta : \mathbb{R}^d \to \mathbb{R}^k$ be a neural network with d input features and k output classes, parameterized by weights θ. The network N_θ assigns the class $i \in \{1, \dots, k\}$ to the point $x \in \mathbb{R}^d$ if $N_\theta(x)_i > N_\theta(x)_j$ for all $j \neq i$.

Let $B_\epsilon(x)$ denote the ℓ_∞-ball of radius ϵ around a point $x \in \mathbb{R}^d$. A network N_θ is called *ϵ-robust* around a point $x \in \mathbb{R}^d$ if N_θ assigns the same class to all points $\tilde{x} \in B_\epsilon(x)$.

More generally, a network N_θ is called *π-robust* around x if it assigns the same class to all points in $\tilde{x} \in \pi(x)$, where $\pi : \mathbb{R}^d \to \mathcal{P}(\mathbb{R}^d)$ describes the capabilities of an attacker. In particular, N_θ is ϵ-robust if it is π-robust for $\pi = B_\epsilon$.

Given a set of labeled training examples $\{(x_i, y_i)\}_{i=1}^n$, the goal of *adversarial training* is to find a θ such that: (i) N_θ assigns the correct class y_i to each example x_i, and (ii) N_θ is π-robust around each example x_i.

Definition 2.1. Given a loss function $L(z, y)$ which is non-negative if $\arg\max_i z_i \neq y$ and strictly negative if $\arg\max_i z_i = y$, we define the *worst-case adversarial loss* L_N on a labeled example (x, y) with respect to network N:

$$L_N(x, y) = \max_{\tilde{x} \in \pi(x)} L(N(\tilde{x}), y).$$

Intuitively, the worst-case adversarial loss is the maximal loss that an attacker can obtain by perturbing the example x to an arbitrary $\tilde{x} \in \pi(x)$. For a given labeled example (x, y), we call a point $\tilde{x} \in \pi(x)$ that maximizes $L(N(\tilde{x}), y)$ a *worst-case adversarial perturbation*.

Using the worst-case adversarial loss, we can formulate adversarial training as the following optimization problem:

$$\min_\theta \max_i L_{N_\theta}(x_i, y_i).$$

If the value of the solution is negative, N_θ classifies all training examples correctly and is π-robust around all points in the training set.

Optimizing Over Sound Approximations It is usually very difficult to find a worst-case adversarial perturbation in the ϵ-ball around an example. Madry et al. (2018) strengthen the network using a heuristic approximation of the worst-case adversarial perturbation and show that the method is practically effective. Kolter & Wong (2017) propose an approach where a superset of the possible classifications for a particular example is determined and the optimization problem is stated in terms of this overapproximation. At a high level, we take a similar approach, but we introduce sound approximations that scale to larger networks and are easier to work with.

Definition 2.2. A sound approximation of a given function $f : \mathbb{R}^d \to \mathbb{R}^k$ under perturbations $\pi : \mathbb{R}^d \to \mathcal{P}(\mathbb{R}^d)$ is a function $\mathcal{A}_{f,\pi} : \mathbb{R}^d \to \mathcal{P}(\mathbb{R}^k)$, such that for every $x \in \mathbb{R}^d$, we have that $f(\pi(x)) \subseteq \mathcal{A}_{f,\pi}(x)$.

To avoid clutter, we usually write $f(S)$ (as in the above definition) as shorthand for the image of $S \subseteq \mathbb{R}^d$ under f.

Sound approximations can be used to prove robustness properties: for example, if we can show that all values $\tilde{z} \in \mathcal{A}_{N,B_\epsilon}(x)$ share the same $\arg\max_i \tilde{z}_i$, then the network N is ϵ-robust around the point x.

Definition 2.3. Given a sound approximation $\mathcal{A}_{N,\pi}$, the *approximate worst-case adversarial loss* $L_N^{\mathcal{A}}$ is given by

$$L_N^{\mathcal{A}}(x, y) = \max_{\tilde{z} \in \mathcal{A}_{N,\pi}(x)} L(\tilde{z}, y).$$

The worst-case adversarial loss L_N can be expressed equivalently as $L_N^{\mathcal{A}}$ using $\mathcal{A}_{N,\pi}(x) = N(\pi(x))$. Therefore, as by definition, we have $N(\pi(x)) \subseteq \mathcal{A}_{N,\pi}(x)$, it follows that $L_N(x, y) \leq L_N^{\mathcal{A}}(x, y)$.

This means that if we choose $\mathcal{A}$ such that: (i) we can compute $L_N^{\mathcal{A}}$, and (ii) we can find θ where $\max_i L_{N_\theta}^{\mathcal{A}}(x_i, y_i)$ is negative, then we have proven the neural network N_θ correct and ϵ-robust for the entire training set.

While the above does not imply that N_θ is ϵ-robust on the test set, we find that approximating the optimal θ in

$$\min_\theta \max_i L_{N_\theta}^{\mathcal{A}}(x_i, y_i)$$

produces networks N_θ that can often be proven robust around previously unseen test examples using the sound approximation $\mathcal{A}$. That is, provable robustness generalizes.

3. Abstract Interpretation

We will approximate neural networks using *abstract interpretation* (Cousot & Cousot, 1977). Abstract interpretation has been recently used to certify robustness of neural networks (Gehr et al., 2018). It has also been used to find constants in small programs using black-box optimization by Chaudhuri et al. (2014), who approximate abstract transformers for a particular probabilistic domain by families of continuous functions. We now introduce the necessary general concepts and our specific instantiations.

Definition 3.1. An *abstract domain* $\mathcal{D}$ is a set equipped with an *abstraction function* $\alpha : \mathcal{P}(\mathbb{R}^p) \to \mathcal{D}$ and a *concretization function* $\gamma : \mathcal{D} \to \mathcal{P}(\mathbb{R}^p)$ for some $p \in \mathbb{N}$.

Intuitively, an element $d \in \mathcal{D}$ corresponds to a set of symbolic constraints over $\mathbb{R}^p$ and $\gamma(d)$ determines the set of points that satisfy the constraints d. The abstraction function α is defined such that $X \subseteq \gamma(\alpha(X))$ for each $X \subseteq \mathbb{R}^p$.

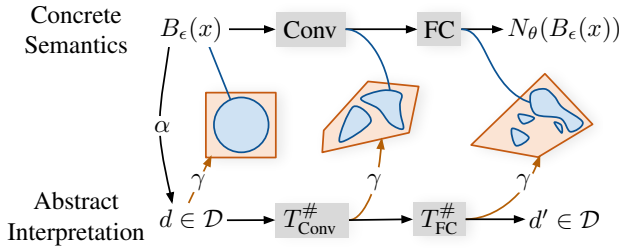


Figure 1. A visualization of abstract interpretation applied on a neural network with convolutional and fully connected layers.

Definition 3.2. A (computable) function $T^\#_f : \mathcal{D} \to \mathcal{D}'$ is called an *abstract transformer* for a function $f : \mathbb{R}^p \to \mathbb{R}^{p'}$ if $f(\gamma(d)) \subseteq \gamma'(T^\#_f(d))$ for all $d \in \mathcal{D}$.

Intuitively, an abstract transformer $T^\#_f$ overapproximates the behavior of a function f on a set $\gamma(d)$ by operating directly on a symbolic representation $d \in \mathcal{D}$ to produce a new abstract element $d' \in \mathcal{D}'$ whose concretization overapproximates the image of f under $\gamma(d)$.

Abstract transformers compose: If $T^\#_f$ and $T^\#_g$ are abstract transformers for functions f and g, then $T^\#_f \circ T^\#_g$ is an abstract transformer for $f \circ g$. Therefore, it suffices to define abstract transformers for each basic operation in a neural network N. If we then write the neural network as a composition of basic operations, we can immediately derive an abstract transformer $T^\#_N$ for the entire N. This abstract transformer $T^\#_N$ induces a sound approximation $\mathcal{A}_{N,\pi}(x) = \gamma(T^\#_N(\alpha(\pi(x))))$ in the sense of Definition 2.2.

We apply abstract interpretation to compute $T^\#_N(\alpha(\pi(x)))$, which describes a superset of the possible outputs of the neural network N under perturbations π. Figure 1 illustrates how abstract interpretation overapproximates the behavior of a network on a concrete set of inputs $\pi(x) = B_\epsilon(x)$. For each intermediate abstract result, γ yields a superset of the corresponding intermediate concrete result (analysis always happens in the abstract, γ is only used to ensure soundness).

We next discuss the abstract domains we use in this work.

Interval Domain The simplest domain we consider is the *interval* (also called Box) domain. Abstract interpretation using this domain is equivalent to computation using standard interval arithmetic. Each element of the domain represents a p-dimensional box, described by its center and deviations in each component. We use this representation for two reasons: (i) it makes the transformations for multiplication and addition efficiently parallelizable on a GPU, and (ii) it exposes essential relationships between our presentation of interval and zonotope domains (discussed below).

An element of the domain is a pair $b = \langle b_C, b_B \rangle$ where $b_C \in \mathbb{R}^p$ is the center of the box, while $b_B \in \mathbb{R}^p_{\geq 0}$ describes (non-negative) deviations. The concretization function γ_I is:

$$\gamma_I(b) = \{b_C + \text{diag}(b_B) \cdot \beta \mid \beta \in [-1, 1]^p\}.$$

Here, $\text{diag}(b_B)$ creates a diagonal $p \times p$ matrix where the entries on the diagonal are those of b_B, and β is an *error vector* used to pick a particular element of the concretization.

Definition 3.3. The *total error* of the i-th component of a box b is $\epsilon_I(b)_i = (b_B)_i$ and the *interval concretization* of the i-th component of b is given by

$$\iota_I(b)_i = [(b_C)_i - \epsilon_I(b)_i,\ (b_C)_i + \epsilon_I(b)_i].$$

Zonotope Domain Interval arithmetic can be imprecise as it does not keep information on how values of variables are related. The *zonotope* domain (Ghorbal et al., 2009) aims to preserve some of these relationships. Unlike the interval domain, where each error term is associated to a particular component, the zonotope domain freely shares error terms among components. In this way, some amount of dependency information can be encoded at moderate costs. The most important feature of the zonotope domain is that there exists an abstract transformer for affine functions (such as the transition function of a fully connected or convolutional layers) which does not lose precision.

An element of the zonotope domain is a pair $z = \langle z_C, z_E \rangle$ where $z_C \in \mathbb{R}^p$ is the center of the zonotope, while $z_E \in \mathbb{R}^{p \times m}$ describes a linear relationship between the error vector $e \in [-1, 1]^m$ and the output components (for arbitrary m). The concretization function γ_Z is given by

$$\gamma_Z(z) = \{z_C + z_E \cdot e \mid e \in [-1, 1]^m\}.$$

Definition 3.4. The *total error* of the i-th component of a zonotope z is $\epsilon_Z(z)_i = \sum_{j=1}^m |(z_E)_{i,j}|$ and the *interval concretization* of the i-th component of z is given by

$$\iota_Z(z)_i = [(z_C)_i - \epsilon_Z(z)_i,\ (z_C)_i + \epsilon_Z(z)_i].$$

The zonotope domain is strictly more expressive than the interval domain: for a box b, its corresponding zonotope z is given by $z_C = b_C$, $z_E = \text{diag}(b_B)$.

Hybrid Zonotope Domain While the zonotope domain is more precise than interval, its transformers are less efficient. The *hybrid zonotope domain*, introduced originally as *perturbed affine arithmetic* by Goubault & Putot (2008), aims to address this issue: its transformers are more accurate than interval, but more efficient than zonotope.

An element of this domain is a triple $h = \langle h_C, h_B, h_E \rangle$ where $h_C \in \mathbb{R}^p$ is the center, $h_B \in \mathbb{R}^p_{\geq 0}$ contains non-negative deviations for each component (similar to interval domain),

and $h_E \in \mathbb{R}^{p \times m}$ describes error coefficients (similar to zonotope domain). The concretization γ_H is given by

$$\gamma_H(h) = \{\widehat{h}(\beta, e) \mid \beta \in [-1,1]^p, e \in [-1,1]^m\},$$

where $\widehat{h}(\beta, e) = h_C + \text{diag}(h_B) \cdot \beta + h_E \cdot e$.

Definition 3.5. The *total error* of the i-th component of a hybrid zonotope h, is $\epsilon_H(h)_i = (h_B)_i + \sum_{j=1}^{m} |(h_E)_{i,j}|$, and the *interval concretization* of the i-th component of h is

$$\iota_H(h)_i = [(h_C)_i - \epsilon_H(h)_i,\ (h_C)_i + \epsilon_H(h)_i].$$

This domain is equally expressive as the zonotope domain but can represent interval constraints more efficiently. Further, the abstract transformers of this domain treat a hybrid zonotope differently than they would treat a zonotope with the same concretization, due to the deviation coefficients.

A box b can be expressed efficiently as a hybrid zonotope h with $h_C = b_C$, $h_B = b_B$ and $m = 0$. A zonotope z can be expressed as a hybrid zonotope h with $h_C = z_C$, $h_B = 0$ and $h_E = z_E$.

4. Abstract Transformers for Zonotope

We now introduce our abstract transformers for the hybrid zonotope domain, specifically ReLU transformers which balance precision with scalability. The transformers are "point-wise": they are efficiently executable on a GPU and can benefit both training and analysis of the network.

There are three basic types of abstract transformers, those that: (i) increase the deviations, (ii) introduce new error terms producing a hybrid zonotope h' with $m' > m$, and (iii) handle deviations and error coefficients separately and do not introduce new error terms. We first discuss the transformers of type (iii). For this type, the transformers of the interval and zonotope domains arise as special cases.

Addition We first consider a function f that replaces the i-th component of the input vector $x \in \mathbb{R}^p$ by the sum of the j-th and k-th components:

$$f(x) = (x_1, \ldots, x_{i-1}, x_j + x_k, x_{i+1}, \ldots, x_p)^T.$$

The corresponding abstract transformer is given by

$$T_f^{\#}(h) = \langle M \cdot h_C, M \cdot h_B, M \cdot h_E \rangle,$$

where the matrix $M \in \mathbb{R}^{p \times p}$ is such that $M \cdot x$ replaces the i-th row of x by the sum of the j-th and k-th rows.

Multiplication Consider a function f that multiplies the i-th component of the input vector $x \in \mathbb{R}^p$ by a known constant $\kappa \in \mathbb{R}$:

$$f(x) = (x_1, \ldots, x_{i-1}, \kappa \cdot x_i, x_{i+1}, \ldots, x_p)^T.$$

The abstract transformer $T_f^{\#}$ for this operation is simple and does not lose any precision. We define

$$T_f^{\#}(h) = \langle M_\kappa \cdot h_C, M_{|\kappa|} \cdot h_B, M_\kappa \cdot h_E \rangle,$$

where $M_\alpha = I + \text{diag}((\alpha - 1) \cdot e_i)$. Here, I is the identity matrix and e_i is the i-th standard basis vector.

Matrix Multiplication and Convolution Consider a function f that multiplies the input vector $x \in \mathbb{R}^p$ by a known matrix $M \in \mathbb{R}^{p' \times p}$:

$$f(x) = M \cdot x.$$

Combining the insights from the addition and multiplication transformers, the abstract transformer $T_f^{\#}$ is given by

$$T_f^{\#}(h) = \langle M \cdot h_C, |M| \cdot h_B, M \cdot h_E \rangle.$$

Here, $|M|$ simply performs component-wise absolute value operation. The above transformer can be easily differentiated and parallelized on the GPU. As convolutions are linear operations, the same approach can be applied for these.

ReLU ReLU is a simple nonlinear activation function:

$$\text{ReLU}(x) = \max(x, 0).$$

In contrast to the abstract transformers discussed so far, there is no single best ReLU abstract transformer for the hybrid zonotope domain. Instead, there are many possible, pairwise incomparable, abstract transformers $T_{\text{ReLU}}^{\#}$.

Abstract domains $\mathcal{D}$ usually support a join operator ($\sqcup$) such that for all abstract elements $d, d' \in \mathcal{D}$, we have

$$\gamma(d) \cup \gamma(d') \subseteq \gamma(d \sqcup d'),$$

and a meet-with-linear-constraint operator ($\sqcap$) such that for all linear constraints $L(x) = x_i < 0$ or $L(x) = x_i \geq 0$ for $1 \leq i \leq p$, we have

$$\gamma(d) \cap \{x \mid L(x)\} \subseteq \gamma(d \sqcap L).$$

In this case, we can define the abstract transformer for ReLU in a general way. Let $f_i(x) = (x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_p)$ be the function that assigns 0 to the i-th component of the input vector. The abstract transformer for ReLU can then be defined as

$$T_{\text{ReLU}}^{\#} = T_{\text{ReLU}_p}^{\#} \circ T_{\text{ReLU}_{p-1}}^{\#} \circ \cdots \circ T_{\text{ReLU}_2}^{\#} \circ T_{\text{ReLU}_1}^{\#},$$

where $T_{\text{ReLU}_i}^{\#}(d) = d \sqcap (x_i \geq 0) \sqcup T_{f_i}^{\#}(d \sqcap (x_i < 0))$.

ReLU_i applies ReLU only to the i-th component of the input vector. While Goubault et al. (2012) demonstrate that accurate join and meet are possible for a hybrid zonotope domain, they take $\Omega(m^3 + m^2 \cdot p)$ time in the worst case, which is significant in the case of deep neural networks. Further, ReLU defined in this way is not parallelizable as the application of ReLU in each *component* is dependent on the application in the previous component.

Point-wise Abstract Transformers for ReLU We define several novel zonotope transformers for the functions ReLU_i. Each modifies the i-th component of the input zonotope and does not depend on any other component (hence, "point-wise"). As a result, the result can be computed in parallel for all components, enabling scalable ReLU analysis.

Definition 4.1 (zBox ReLU transformer)**.** For a zonotope z and index i, we define $z' = T^{\#(\text{zBox})}_{\text{ReLU}_i}(z)$ with $m' = m + 1$. If $\min(\iota_i(z)) \geq 0$, then z' is z with an additional unused error term, meaning new entries in z'_E are set to 0. Otherwise

$$\begin{aligned}(z'_X)_t &= (z_X)_t, \text{ for } X \in \{C, E\}, t \neq i,\\ (z'_C)_i &= \text{ReLU}(\tfrac{1}{2}\max(\iota(z)_i)),\\ (z'_E)_{i,l} &= 0, \text{ for } l \leq m,\\ (z'_E)_{i,m+1} &= \text{ReLU}(\tfrac{1}{2}\max(\iota(z)_i)),\\ (z'_E)_{j,m+1} &= 0, \text{ for } j \neq i.\end{aligned}$$

$T^{\#(\text{zBox})}_{\text{ReLU}_i}(z)$ propagates the input zonotope unchanged if it can prove the i-th component is non-negative (then ReLU_i has no effect). Otherwise, it bounds the i-th component of the output by a suitable independent interval using the new error term. In both cases, the number of error terms in z' is the same, allowing for more effective parallelization.

We also define a transformer which uses the above transformer with incomparable precision.

Definition 4.2 (zDiag ReLU transformer)**.** Given a zonotope z and index i, we define a ReLU abstract transformer $z' = T^{\#(\text{zDiag})}_{\text{ReLU}_i}(z)$ where $m' = m + 1$. If the condition $\min(\iota(z)_i) < 0 < \max(\iota(z)_i)$ holds, we have

$$\begin{aligned}(z'_X)_t &= (z_X)_t \text{ for } X \in \{C, E\}, t \neq i,\\ (z'_C)_i &= (z_C)_i - \tfrac{1}{2}\min(\iota(z)_i),\\ (z'_E)_{i,l} &= (z_E)_{i,l} \text{ for } l \leq m,\\ (z'_E)_{i,m+1} &= -\tfrac{1}{2}\min(\iota(z)_i),\\ (z'_E)_{j,m+1} &= 0, \text{ for } j \neq i.\end{aligned}$$

Otherwise, $z' = T^{\#(\text{zBox})}_{\text{ReLU}_i}(z)$.

Finally, we define two transformers which combine zBox and zDiag in different ways.

Definition 4.3 (zSwitch ReLU transformer)**.** Transformer $T^{\#(\text{zSwitch})}_{\text{ReLU}_i}$ uses $T^{\#(\text{zBox})}_{\text{ReLU}_i}$ if $|\min(\iota(z)_i)| > |\max(\iota(z)_i)|$. Otherwise, it uses $T^{\#(\text{zDiag})}_{\text{ReLU}_i}$.

Definition 4.4 (zSmooth ReLU transformer)**.** Transformer $T^{\#(\text{zSmooth})}_{\text{ReLU}_i}$ takes a weighted average of the results of $T^{\#(\text{zBox})}_{\text{ReLU}_i}$ and $T^{\#(\text{zDiag})}_{\text{ReLU}_i}$, with weights $|\min(\iota(z)_i)|$ and $|\max(\iota(z)_i)|$ respectively.

Point-wise ReLU for Hybrid Zonotopes The hybrid zonotope transformers (hSwitch, hSmooth) operate the same way as the respective zonotope versions (zSwitch, zSmooth), but they do not add new error terms. Instead, they accumulate the computed error in the i-th component of h_B.

5. Adversarial Training

We now introduce adversarial training with our domains.

5.1. Approximate Worst-Case Adversarial Loss

Let the loss $L(z, y) = \max_{y' \neq y}(z_{y'} - z_y)$. This loss satisfies the requirements for adversarial training from Definition 2.1. Let us define our approximate worst-case adversarial loss by instantiating $\mathcal{A}_{N,\pi}(x)$ from Definition 2.3 as

$$\mathcal{A}_{N,\pi}(x) = \gamma(T^{\#}_N(\alpha(\pi(x)))).$$

This means that we take the region $\pi(x)$, abstract it so it can be captured in our abstract domain, obtain $\alpha(\pi(x))$, and then apply the neural network transformer $T^{\#}_N$ (as discussed so far) to that result. For $\pi = B_\epsilon$, the expression $\alpha(\pi(x))$ corresponds to a simple interval constraint that can be computed easily for each of the discussed abstract domains. Once we obtain the abstract output of the transformer, we can apply γ to obtain all concrete output points represented by this result. With this instantiation we obtain the loss

$$L^{\mathcal{A}}_N(x, y) = \max_{\tilde{z} \in \gamma(T^{\#}_N(\alpha(\pi(x))))} L(\tilde{z}, y).$$

This is the approximate worst-case adversarial loss when we use abstract interpretation for approximation. We can compute the loss $L^{\mathcal{A}}_N(x, y)$ using

$$L^{\mathcal{A}}_N(x, y) = \max_{y' \neq y}(\max \iota(T^{\#}_{f_{y'}}(T^{\#}_N(\alpha(\pi(x)))))),$$

provided the following conditions hold: (i) an interval concretization function ι exists for which we can compute interval upper bounds $\max \iota(x)$, (ii) we can compute $\alpha(\pi(x_i))$, and (iii) the linear function $f_{y'}(z) = z_{y'} - z_y$ has a precise abstract transformer $T^{\#}_{f_{y'}}$. All three of these conditions hold for the interval, zonotope and hybrid zonotope domains.

5.2. Line Segments as (Hybrid) Zonotopes

We show how to consider input regions in other shapes beyond the standard ℓ_∞-balls used in local robustness properties. We use the observation that when two points with the same classification in the training set are close enough, the points on the line between them should be classified the same. Figure 2 shows an intuitive illustration. Black dots represent data and grey regions represent the ground truth classifications. The network

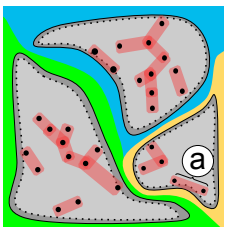


Figure 2. Visualizing line segment training with 3 classifications.

Dataset	Model	Type	# Hidden Units	# Parameters	Train Time (s/epoch)		Total Testing Time (s)	
					Baseline	Box	Box	hSwitch
MNIST	FFNN	fully connected	510	119910	0.610	2.964	0.076	0.184
	ConvSmall	convolutional	3604	89606	0.560	4.014	0.056	0.360
	ConvBig	convolutional	34688	893418	1.839	7.229	0.060	7.431
	ConvSuper	convolutional	88500	10985962	4.391	15.743	0.080	12.856
	Skip	residual	71600	6301890	3.703	12.613	0.073	11.313
CIFAR10	FFNN	fully connected	510	348710	1.273	4.145	0.066	1.018
	ConvSmall	convolutional	4852	125318	0.718	3.979	0.065	1.870
	ConvMed	convolutional	6244	214918	1.462	5.200	0.051	1.953
	ConvBig	convolutional	62464	2466858	6.585	21.539	0.062	11.372
	ConvSuper	convolutional	124128	16229418	23.416	74.247	0.089	40.270
	Skip	residual	97730	8760802	14.245	42.482	0.083	25.198

Table 1. A table showing the size of our networks, the time it takes to train one epoch (averaged over 200 epochs), and the best total testing time for 500 samples, with the maximum batch size allowable by the GPU memory for each domain and network combination. The testing times are for a baseline-trained network, the times for a Box-trained network are similar.

learns to classify points in the yellow, green, and blue regions. Data points only actually appear within a smaller shell within the class boundary (dotted lines). The red lines connecting the points show how these points are grouped into line abstractions. Marker "a" shows a bad scenario, where one of the abstractions comes near the ground truth class boundary. We experimented with encoding the segment between nearby points instead of simple robustness regions, aiming to improve the result of training. In terms of encoding, with a zonotope, we can conveniently represent a line segment between two points x and y as follows:

$$(z_C)_i = \frac{1}{2}(x_i + y_i),\ (z_E)_{i,1} = \frac{1}{2}\left|x_i - y_i\right|.$$

For hybrid zonotopes, we additionally consider a width parameter w:

$$(h_C)_i = \frac{1}{2}(x_i + y_i),\ (h_B)_i = w,\ (h_E)_{i,1} = \frac{1}{2}\left|x_i - y_i\right|.$$

6. Experimental Evaluation

We implemented our approach in a system called DIFFAI[2] and evaluated it extensively across a range of datasets and network architectures. We demonstrate that DIFFAI training scales to networks larger than those of prior work and that networks trained with DIFFAI are more provably robust than those trained with state-of-the-art defenses.

6.1. Experimental Setup

Our system is built on top of PyTorch (Paszke et al., 2017). For all of our experiments, we used the Adam Optimizer (Kingma & Ba, 2014), with the default parameters and a learning rate (*lr*) of 0.0001, unless otherwise specified. Additionally, we used norm-clipping on the weights after every batch with a max-norm of $10,000$. For training we use a batch size of 500. For testing, batch sizes vary from network to network depending on the experiment and the GPU used. We ran on a GeForce GTX 1080 Ti, and a K80. For all comparisons on a single network and dataset (shown in the tables), we used the same initial weights and presented the models with the same batches. In all experiments, accuracy and provability is tested on 500 samples.

We evaluate DIFFAI on four different datasets: MNIST, CIFAR10, FashionMNIST (F-MNIST) and SVHN. We also consider networks of various sizes, shown in Table 1 (sizes for SVHN are the same as CIFAR10 and sizes for Fashion-MNIST are the same as MNIST).

Training While we did find that it was often possible to train entirely using the loss function described in Section 5.1, we achieved better results by combining it with a standard cross-entropy loss function during training. Furthermore, we found that the loss in Section 5.1 continued to provide gradient information to the optimizer long after the example was proven correct, which was detrimental to learning for other examples. We found that applying a smooth version of ReLU – softplus (Dugas et al., 2001) – to this loss helped to avoid this issue. Our combined loss is therefore

$$\mathcal{L}_\theta(x, y) = \lambda \cdot \text{softplus}(L^{\mathcal{A}}_{N_\theta}(x, y)) + H(\text{softmax}(N_\theta(x)), y),$$

where λ is set to 0.1 in all of our experiments (except line training). The total loss for a batch is obtained by *adding* the losses for all examples in the batch.

[2] Available at: `http://diffai.ethz.ch`

Dataset	ϵ	lr	Model	Train Method	Total Train Time (s)	Test Error %	Lower Bound % PGD	Upper Bound % Box	Upper Bound % hSwitch
MNIST	0.1	10^{-3}	ConvBig	Baseline	367.80	0.8	3.0	100.0	100.0
				PGD	1847.76	0.2	1.6	100.0	99.8
				Box	1445.76	1.0	2.4	14.0	3.4
			ConvSuper	Baseline	878.28	1.6	2.4	100.0	97.2
				PGD	4867.56	1.2	1.6	100.0	88.8
				Box	3148.68	1.0	2.8	11.8	3.6
			Skip	Baseline	731.40	1.4	3.8	100.0	100.0
				PGD	3935.04	1.0	2.0	100.0	83.4
				Box	2734.44	1.6	4.4	13.6	5.8
CIFAR10	0.007	10^{-4}	ConvBig	Baseline	1317.00	32.4	36.2	100.0	100.0
				PGD	8574.72	31.4	35.8	100.0	100.0
				Box	4307.88	55.0	58.6	76.4	61.4
			ConvSuper	Baseline	4683.24	37.4	42.4	100.0	100.0
				PGD	29828.52	35.4	41.0	100.0	100.0
				Box	14849.40	52.8	59.2	83.8	64.2
			Skip	Baseline	802.92	36.8	41.8	100.0	88.0
				PGD	4828.68	33.6	40.2	100.0	82.8
				Box	2980.92	38.0	45.4	72.2	47.8

Table 2. Results on time, test error, and adversarial bounds after 200 epochs with L_2 regularization constant of 0.01 and 5 PGD iterations.

6.2. Comparing Against Prior Defenses and Analyzers

We evaluated the performance of DIFFAI using ℓ_∞-balls and Box training against standard baseline training. We also trained with the state-of-the-art defense of Madry et al. (2018), which permutes each batch using the untargeted PGD attack. The *adversarial test error* is the largest error an adversary can achieve by perturbing all examples in the test set. As we cannot efficiently compute the adversarial test error, we instead give lower and upper bounds using the PGD attack and DIFFAI respectively. Some of our results are shown in Table 2 (all results are in the supplementary).

Scalability of Training To our knowledge, we analyzed and defended the largest networks considered so far in the context of provable robustness, in terms of both number of neurons and weights. As shown in Table 1, we were able to train a network (ConvSuper on CIFAR10) with 124000 neurons and over 16 million weights in under 75 seconds per epoch for a total time of less than 5 hours. The net trained is larger than the largest considered by Kolter & Wong (2017), who took 10 hours to train a significantly smaller network, and do not report stand-alone testing speed. Often, DIFFAI's Box training is even faster than PGD training with 5 iterations. For ConvSuper on CIFAR10 in Table 2, Box took under 4.5 hours to train, while PGD took over 8 hours.

Scalability of Testing DIFFAI can also analyze large networks: for a given example, it can verify ConvSuper on CIFAR10 in under 2×10^{-4} seconds with Box and 0.1 seconds with hSwitch. This is an order of magnitude speed-up over the current state-of-the-art (Gehr et al., 2018).

Applicability to Complex Nets We also trained and tested a network, Skip, with a residual connection (He et al., 2016) for all datasets. Scalability results for MNIST and CIFAR10 are shown in Table 1, and the adversarial performance of Box training is shown in Table 2. While Skip is quite wide, it is only 5 layers deep with only one residual connection, using concatenation instead of addition.

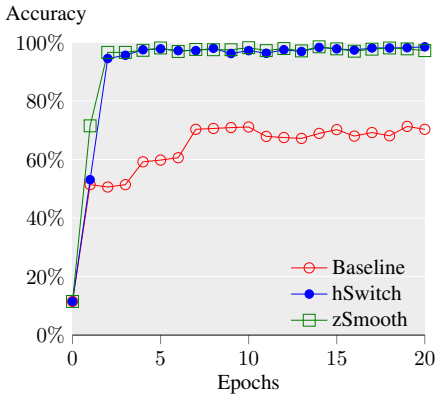
Provability PGD defense tends to slightly improve accuracy over baseline and those networks are typically less attackable by a PGD attack than box-trained ones. However, box-trained networks are more provably robust (via Box or hSwitch) than PGD-defended networks. Table 2 also shows that Box training produces more provably robust networks than baseline and often, with little loss of accuracy.

DIFFAI achieved consistently below 4% test error on the MNIST benchmark for convolutional networks, as can be seen in Table 2 (and in supplementary). When trained using Box, ConvSuper achieves 1% test error and DIFFAI can prove an upper bound of 3.6% on the adversarial test error, close to the lower bound of 2.8% given by PGD. In contrast, baseline training produced a network which is less accurate and could not be proved robust for any test example.

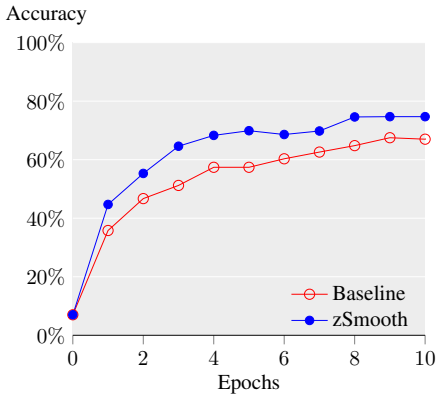
In both Table 2 and Table 3, hSwitch and zSwitch always produce better upper bounds than Box. In theory, these domains are incomparable to Box, however, in practice,

Dataset	ϵ	Epochs	Model	Train Method	Train Time (s)	Test Error %	Lower Bound % PGD	Upper Bound % Box	Upper Bound % zSwitch
F-MNIST	0.1	200	FFNN	Baseline	119	5.4	98.8	100.0	100.0
				Box	608	91.4	91.4	100.0	100.0
				hSmooth	4316	15.6	71.8	100.0	79.0
CIFAR10	0.03	20	ConvSmall	Baseline	572	35.0	54.8	100.0	83.8
				Box	999	44.2	56.4	77.6	63.0
				hSmooth	36493	38.0	53.6	99.8	62.6
SVHN	0.01	20	ConvSmall	Baseline	700	15.8	83.6	100.0	98.0
				Box	1223	27.0	78.4	92.6	89.8
				hSmooth	43859	19.6	78.4	98.4	89.0

Table 3. Results showing that training with hSmooth can lead to improved accuracy (and upper bounds) over training with Box.



(a) MNIST with $w = 0.05$



(b) SVHN with $w = 0.001$

Figure 3. Accuracy of segment training on FFNN. No regularization and learning rate of 0.001. (a) Batch size 200 for Baseline and 20 for hSwitch and zSmooth; λ of 10^5. (b) Batch size 150 for Baseline and 30 for zSmooth. λ scheduled with a power of 10 over 10^5 examples starting at 10^{-6} and ending at 10^{-4}.

they are typically significantly more precise. As testing with Box is essentially free, and hSwitch is quite efficient, we suggest testing with both and selecting the lower value.

Training with Accurate Domains Occasionally, training with Box led to much lower accuracy. In these cases, we attempted to improve the accuracy by instead training with the more accurate (and more expensive) hSmooth domain. Results can be seen in Table 3, where for example the FFNN network for F-MNIST has 91.4% testing error when trained with Box and 15.6% error when trained with hSmooth.

As training with hSmooth is significantly less space efficient and cannot be done with batches on convolutional networks, a smaller network was used, ConvSmall, to demonstrate this point. In CIFAR10 for example, hSmooth produced a network nearly as accurate as ConvSuper trained using baseline. This was accomplished with little reduction to provable robustness (when testing with zSwitch).

6.3. Segment Training for Higher Accuracy

To test DIFFAI on more complex abstract regions, we trained with line segments connecting examples, as described in Section 5.2. For every element in a batch, we built a zonotope connecting it to the nearest (in terms of ℓ_2 distance) other element in the batch with the same class.

The plot in Figure 3 demonstrates that line segment training improves accuracy. After 10 epochs and 1.5 hours, hSwitch with line segment training reached the highest accuracy of 74.7% for SVHN. After 20 epochs and 10 hours, zSwitch and hSwitch both reached an accuracy of 97.4% for MNIST, significantly higher than the 70% achieved by the baseline.

7. Conclusion

We showed how to apply abstract interpretation for defending neural networks against adversarial perturbations and introduced several zonotope transformers which carefully balance precision with scalability. Our results indicate the training approach scales to networks larger than those of prior work and the resulting networks are more provably robust than networks trained with state-of-the-art defenses.

References

Athalye, A. and Sutskever, I. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*, 2017.

Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., and Rival, X. A static analyzer for large safety-critical software. In *Programming Language Design and Implementation (PLDI)*, 2003.

Carlini, N., Katz, G., Barrett, C., and Dill, D. L. Ground-truth adversarial examples. *arXiv preprint arXiv:1709.10207*, 2017.

Chaudhuri, S., Clochard, M., and Solar-Lezama, A. Bridging boolean and quantitative synthesis using smoothed proof search. In *Symposium on Principles of Programming Languages (POPL)*, 2014.

Cousot, P. and Cousot, R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Symposium on Principles of Programming Languages (POPL)*, 1977.

Dugas, C., Bengio, Y., Bélisle, F., Nadeau, C., and Garcia, R. Incorporating second-order functional knowledge for better option pricing. In *Advances in Neural Information Processing Systems (NIPS)*, 2001.

Evtimov, I., Eykholt, K., Fernandes, E., Kohno, T., Li, B., Prakash, A., Rahmati, A., and Song, D. Robust physical-world attacks on deep learning models. *arXiv preprint arXiv:1707.08945*, 2017.

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

Ghorbal, K., Goubault, E., and Putot, S. The zonotope abstract domain taylor1+. In *International Conference on Computer Aided Verification (CAV)*, 2009.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Goubault, E. and Putot, S. Perturbed affine arithmetic for invariant computation in numerical program analysis. *arXiv preprint arXiv:0807.2961*, 2008.

Goubault, E., Le Gall, T., and Putot, S. An accurate join for zonotopes, preserving affine input/output relations. *Electronic Notes in Theoretical Computer Science*, 2012.

Gu, S. and Rigazio, L. Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068*, 2014.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kolter, J. Z. and Wong, E. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 2017.

Krizhevsky, A. Learning multiple layers of features from tiny images. 2009.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. 2018.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Papernot, N., McDaniel, P., and Goodfellow, I. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.

Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. Practical black-box attacks against machine learning. In *Asia Conference on Computer and Communications Security*. ACM, 2017.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017.

Raghunathan, A., Steinhardt, J., and Liang, P. Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*, 2018.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

A. Neural Networks Evaluated

We train six networks: one feed forward, four convolutional (without maxpool), and one with a residual connection. In the following descriptions, we use $\text{Conv}_s C \times W \times H$ to mean a convolutional layer that ouputs C channels, with a kernel width of W pixels and height of H, with a stride of s which then applies ReLU to every output. FC n is a fully connected layer which outputs n neurons without automatically applying ReLU.

FFNN A 5 layer feed forward net with 100 nodes in each and a ReLU after each layer. This network has a ReLU after the last layer.

ConvSmall Our smallest convolutional network with no convolutional padding.

$$x \to \text{Conv}_2 16 \times 4 \times 4 \to \text{Conv}_2 32 \times 4 \times 4 \to \text{FC } 100 \to z.$$

ConvMed The same as ConvSmall, but with a convolutional padding of 1.

$$x \to \text{Conv}_2 16 \times 4 \times 4 \to \text{Conv}_2 32 \times 4 \times 4 \to \text{FC } 100 \to z.$$

ConvBig A significantly larger convolutional network with a convolutional padding of 1.

$$\begin{aligned} x &\to \text{Conv}_1 32 \times 3 \times 3 \to \text{Conv}_2 32 \times 4 \times 4 \\ &\to \text{Conv}_1 64 \times 3 \times 3 \to \text{Conv}_2 64 \times 4 \times 4 \\ &\to \text{FC } 512 \to \text{ReLU} \to \text{FC } 512 \to z. \end{aligned}$$

ConvSuper Our largest convolutional network with no padding.

$$\begin{aligned} x &\to \text{Conv}_1 32 \times 3 \times 3 \to \text{Conv}_1 32 \times 4 \times 4 \\ &\to \text{Conv}_1 64 \times 3 \times 3 \to \text{Conv}_1 64 \times 4 \times 4 \\ &\to \text{FC } 512 \to \text{ReLU} \to \text{FC } 512 \to z. \end{aligned}$$

Skip Two convolutional networks of different sizes, which are then concatenated together. This network uses no convolutional padding.

$$\begin{aligned} x &\to \text{Conv}_1 16 \times 3 \times 3 \\ &\to \text{Conv}_1 16 \times 3 \times 3 \\ &\to \text{Conv}_1 32 \times 3 \times 3 \to \text{FC } 200 \to o_1, \\ x &\to \text{Conv}_1 32 \times 4 \times 4 \\ &\to \text{Conv}_1 32 \times 4 \times 4 \to \text{FC } 200 \to o_2, \\ \text{CAT}(o_1, o_2) &\to \text{ReLU} \to \text{FC } 200 \to \text{ReLU} \to z. \end{aligned}$$

B. Training Results

For all experiments (except with segment training), we trained with an L_2 regularization constant of 0.01, and a λ of 0.1. We halt training after 200 epochs. For MNIST, we used a learning rate of 10^{-3}. For all other experiments, the learning rate was 10^{-4}. For both testing and training, we used the untargeted PGD attack with $k = 5$ iterations. At testing time, we used 500 examples.

Name	Source	Ex/Epoch	Inp Dim
MNIST	LeCun et al. (1998)	60000	$1 \times 28 \times 28$
F-MNIST	Xiao et al. (2017)	60000	$1 \times 28 \times 28$
CIFAR10	Krizhevsky (2009)	50000	$3 \times 32 \times 32$
SVHN	Netzer et al. (2011)	73257	$3 \times 32 \times 32$

Table 4. The datasets we evaluate with. All use 10 classifications.

				Lower Bound %	Upper Bound %	
Model	Train Method	Total Train Time (s)	Test Error %	PGD	Box	hSwitch
FFNN	Baseline	121.92	2.4	40.8	100.0	100.0
	PGD	368.76	1.0	3.8	100.0	100.0
	Box	592.80	5.6	13.6	33.0	53.0
ConvSmall	Baseline	123.36	1.2	2.4	100.0	49.8
	PGD	515.64	0.8	1.8	100.0	22.2
	Box	690	2.4	4.4	17.8	5.8
ConvBig	Baseline	367.80	0.8	3.0	100.0	100.0
	PGD	1847.76	0.2	1.6	100.0	99.8
	Box	1445.76	1.0	2.4	14.0	3.4
ConvSuper	Baseline	878.28	1.6	2.4	100.0	97.2
	PGD	4867.56	1.2	1.6	100.0	88.8
	Box	3148.68	1.0	2.8	11.8	3.6
Skip	Baseline	731.40	1.4	3.8	100.0	100.0
	PGD	3935.04	1.0	2.0	100.0	83.4
	Box	2734.44	1.6	4.4	13.6	5.8

Table 5. MNIST with $\epsilon = 0.1$

				Lower Bound %	Upper Bound %	
Model	Train Method	Total Train Time (s)	Test Error %	PGD	Box	hSwitch
FFNN	Baseline	120.84	3.0	99.0	100.0	100.0
	PGD	357.48	2.0	10.2	100.0	100.0
	Box	570.84	13.4	50.6	77.8	84.4
ConvSmall	Baseline	124.44	1.6	20.2	100.0	100.0
	PGD	518.64	1.8	4.6	100.0	100.0
	Box	678.36	3.2	9.0	28.2	19.4
ConvBig	Baseline	368.76	2.4	18.2	100.0	100.0
	PGD	1863.12	1.6	4.0	100.0	100.0
	Box	1436.40	3.4	6.2	23.2	18.0
ConvSuper	Baseline	895.56	1.2	15.0	100.0	100.0
	PGD	5021.28	1.0	1.0	100.0	100.0
	Box	3216.72	2.8	8.0	19.0	23.0

Table 6. MNIST with $\epsilon = 0.3$

				Lower Bound %	Upper Bound %	
Model	Train Method	Total Train Time (s)	Test Error %	PGD	Box	hSwitch
FFNN	Baseline	254.52	53.4	73.8	100.0	100.0
	PGD	956.64	45.4	53.6	100.0	100.0
	Box	829.08	48.6	57.8	84.4	65.4
ConvMed	Baseline	292.44	40.2	44.0	100.0	54.0
	PGD	1472.16	38.8	43.6	100.0	52.4
	Box	1040.04	42.8	46.4	74.6	47.8
ConvBig	Baseline	1317.00	32.4	36.2	100.0	100.0
	PGD	8574.72	31.4	35.8	100.0	100.0
	Box	4307.88	55.0	58.6	76.4	61.4
ConvSuper	Baseline	4683.24	37.4	42.4	100.0	100.0
	PGD	29828.52	35.4	41.0	100.0	100.0
	Box	14849.40	52.8	59.2	83.8	64.2
Skip	Baseline	802.92	36.8	41.8	100.0	88.0
	PGD	4828.68	33.6	40.2	100.0	82.8
	Box	2980.92	38.0	45.4	72.2	47.8

Table 7. CIFAR10 with $\epsilon = 0.007$

				Lower Bound %	Upper Bound %	
Model	Train Method	Total Train Time (s)	Test Error %	PGD	Box	hSwitch
FFNN	Baseline	263.16	57.6	96.8	100.0	100.0
	PGD	1000.68	46.6	46.6	61.4	100.0
	Box	849.00	52.2	68.8	90.0	82.6
ConvMed	Baseline	98.20	40.4	61.8	100.0	100.0
	PGD	1546.92	42.2	54.8	100.0	97.8
	Box	1061.52	45.8	60.0	85.8	64.8
ConvBig	Baseline	1329.48	37.2	61.6	100.0	100.0
	PGD	8733.84	41.6	56.2	100.0	100.0
	Box	4355.76	51.6	61.4	83.2	75.8
ConvSuper	Baseline	4724.76	39.8	66.2	100.0	100.0
	PGD	31140.72	39.6	56.2	100.0	100.0
	Box	15314.76	54.2	64.6	83.8	87.6
Skip	Baseline	805.32	39.6	69.4	100.0	100.0
	PGD	4916.04	37.0	54.0	100.0	100.0
	Box	2789.52	52.6	68.6	90.8	78.0

Table 8. CIFAR10 with $\epsilon = 0.03$

				Lower Bound %	Upper Bound %	
Model	Train Method	Total Train Time (s)	Test Error %	PGD	Box	hSwitch
FFNN	Baseline	151.50	22.6	13.8	100.0	93.8
	PGD	459.61	19.2	24.0	100.0	88.6
	Box	722.46	43.8	10.0	66.2	32.2
ConvMed	Baseline	144.32	15.0	19.8	100.0	54.8
	PGD	624.59	14.0	4.4	100.0	40.0
	Box	838.21	23.6	9.0	66.2	32.2
ConvBig	Baseline	533.31	11.6	9.6	100.0	98.4
	PGD	2745.82	14.0	3.6	100.0	96.4
	Box	2065.11	22.2	7.4	57.4	20.8
Skip	Baseline	969.48	13.8	10.4	100.0	96.8
	PGD	5844.00	13.6	5.0	100.0	86.8
	Box	3352.24	25.0	6.6	53.6	23.0

Table 9. SVHN with $\epsilon = 0.01$

				Lower Bound %	Upper Bound %	
Model	Train Method	Total Train Time (s)	Test Error %	PGD	Box	hSwitch
FFNN	Baseline	121.08	3.4	99.2	100.0	100.0
	PGD	345.24	3.6	13.8	100.0	100.0
	Box	568.68	99.1	91.0	100.0	100.0
ConvMed	Baseline	123.36	1.6	61.6	100.0	100.0
	PGD	507.84	1.0	5.4	100.0	100.0
	Box	668.04	3.4	14.8	28.6	28.2
ConvBig	Baseline	368.40	1.6	48	100.0	100.0
	PGD	1819.08	1.2	3.0	100.0	100.0
	Box	1429.56	1.8	4.2	11.8	22.0
Skip	Baseline	687.72	1.4	58.6	100.0	100.0
	PGD	3791.76	1.2	3.4	100.0	100.0
	Box	2310.12	3.2	10.0	20.6	29.4

Table 10. F-MNIST with $\epsilon = 0.1$